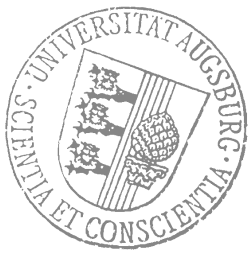
High-Performance Computing with Graphics Processing Units for Physics of Complex Systems

Dissertation

Institut für Physik

Mathematisch-Naturwissenschaftliche Fakultät

Universität Augsburg



eingereicht von

Armin Seibert

Augsburg, Dezember 2013

Erstgutachter: Prof. Dr. Peter Hänggi

Zweitgutachter: Prof. Dr. Liviu Chioncel

Tag der mündlichen Prüfung: 17.Januar 2014

Für Dagmar

Contents

1. Introduction and historical overview

The interest of a physicist in computational power is twofold. On the one hand, the set of physical problems that can be solved numerically is much larger than that of analytical problems. On the other hand, computers themselves can be viewed as physical systems which perform physical processes. So the idea of computation has finally been reversed by preparing a system in such a way that its physical evolution tells us a certain result. This shall finally be the case of a quantum computer. The most astonishing relation in the context of computation is the exponential growth of computational power according to Moore's law (Fig. 1.1). The computational power of the chip in a modern washing machine is said to be more powerful than the computer used for the Apollo 11 in 1969.

In 1936 Alan Turing coined what is now regarded as the Turing completeness. The invention of the first Turing complete computer can be dated back to the Zuse 3 in 1941 or the ENIAC in 1944. One could argue that such earth-shattering ideas only come up under huge pressure, like that of a war. But this invention was rather an incident. The further electrification of the Zuse 3 was even rejected as 'not war important'. Since this phase transition during the Second World War the calculation speed of supercomputers enjoys exponential growth.

But no exponential lasts forever. So the clock speed of a single processor has settled at around 3 GHz almost ten years ago [Sut05]. The reason for this seems to be a very physical one, the heat in the integrated circuits. Nevertheless, computational power per € and the size of transistors still seem to follow Moore's law. The only difference is that computers consist of many processors.

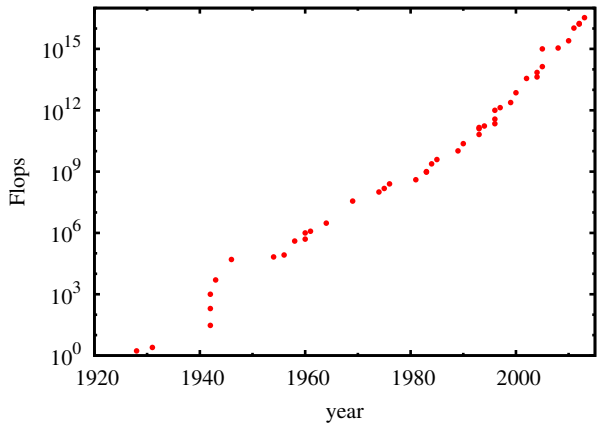


Figure 1.1.: Computational power of the world's fastest supercomputer (data from [Sup]).

Thus if we want to profit from increasing computational power, we must write parallel programs. The need for parallelization is obvious for improvement of numerical schemes. Therefore the issue of parallelization has gained attention in industry, gaming and science. The latter profits most from the other ones in this respect.

Measuring computational power was just a question of clock speed for a long time. This is not the whole story since several operations can be done during one clock cycle and other facets of the architecture play a role, like pipelines and caches. In this age of parallel computing the architecture has become more diversified and a single number to measure performance is hard to get. It depends very much on the purpose. In this thesis we will only refer to the number of Flops (Floating point operations per second) which is widely accepted, e.g. the Flops on execution of the LINPACK benchmark are used to rank the fastest supercomputers of the world.

To describe the dependence of the running time T of an algorithm on the problem size N the notation $\mathcal{O}(N^\alpha)$ is used. It means that for large numbers N the algorithm will take time $T \propto N^\alpha$. This allows to estimate roughly the scaling of the computational scheme via the computation of several small size problems. On top, it also extrapolates the value of an algorithm to the future. The constant prefactor is certainly relevant but is omitted since it carries less

information. The Mergesort algorithm for sorting numbers, e.g., is $\mathcal{O}(N \log N)$. Still the standard algorithm of choice is the Quicksort algorithm [Hoa62] which is $\mathcal{O}(N^2)$. This $\mathcal{O}(N^2)$ is only valid for the worst case. On average Quicksort also needs only $\mathcal{O}(N \log N)$ with a better prefactor than merge sort. Another number that will play a crucial role in the future is the ability of an algorithm to be distributed on independent tasks. If the speedup S is proportional to the number of computer nodes n, the algorithm is optimal and has a future. If the algorithm is serial, i.e. no single operation is calculated in parallel, or the speedup drops fast with the number of nodes, it can be incredibly fast, but it will not take advantage of parallelization. Generally the speedup for a code where a fraction r_p is parallelizable would follow Amdahl's law [Amd67]:

$$S = \frac{1}{(1 - r_p) + \frac{r_p}{n}}. \tag{1.1}$$

A question left at this point is what hardware and software to use. The software that accounts for parallelization is called an application programming interface (API). For most programmers the relevant options are very few at the moment.

- OpenMP: One can use multiple CPUs on one motherboard and the OpenMP as API. This choice is easy to handle. One is limited by the number of processors on the motherboard, which is around 32 at the moment. This limits the computational power to 400 GFlops.

- MPI: The Message Passing Interface (MPI) is able to distribute jobs over a cluster of computers. So the number of processors is theoretically unlimited. The drawback is the bandwidth. One has to use slow network cables to pass the data between the processors.

- CUDA: Compute Unified Device Architecture (CUDA) is used for execution on Nvidia graphics processing units (GPU). Several GPUs can be addressed on one computer. The latest GPU has 2500 cores with 1.3 TFlops. Eight GPUs can be used per motherboard. So the performance of a CUDA computer would be 10 TFlops.

A noncommercial alternative for general purpose computing on graphics processing units (GPGPU) is the OpenCL language which runs also on ATI graphic cards. The latest version is from November 2011 and it doesn't seem to attract much attention any more. A Hardware alternative comes from Intel who jumped on the GPU bandwagon in the middle of 2012 with its Xeon Phi (Larrabee) GPU. This GPU is a collection of around 60 Xeon cores. They can be run via MPI [Int13]. The memory has the same size as standard GPUs and the performance is with 600 GFlops half as fast as the latest Nvidia GPU [Don13]. Nevertheless the fastest Supercomputer of today is the Phi based Tianhe-2 in Guangzhou, China. The fastest GPU based supercomputer is the Titan built of 18688 AMD Opteron 6274 16-core CPUs and as many Nvidia Tesla K20X GPUs in November 2012 in the Oak Ridge National Laboratory, USA with a performance of 17.5 PFlops.

We decided to choose CUDA. The reasons for this are given in the next chapter including an introduction to CUDA. In the subsequent chapter the Arnold diffusion is investigated, an old and basic problem of Hamiltonian chaos that is still not well understood but fits perfectly for numerical treatment on GPUs.

2. CUDA computing with GPU: General introduction

Graphics processing units (GPU) have grown so big that they alone have become the brain of the modern computer. They can perform most of the calculations and consume most of the power. A recent GPU for scientific computations is shown on Fig. 2.1. The term "graphics card" is only historically justified: The technology stems from the gaming graphics cards and they occupy the same port on the motherboard (PCIe) the bandwidth of which has been thoroughly increased in the last 15 years. But a scientific GPU has no ports for a screen any more and it is built for general purpose computing solely.

In this chapter the C/C++ experienced programmer is briefly introduced to the features of GPU hardware and the CUDA language to program it. If a "pointer pointer" is already perceived as something difficult to handle, it must be mentioned that CUDA offers features like "mapped pinned host

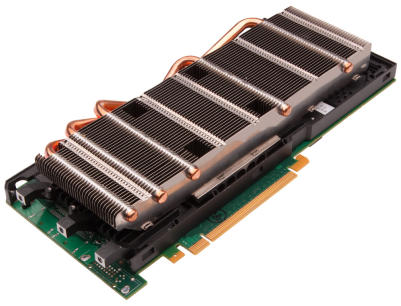


Figure 2.1.: Nvidia Tesla M2090 GPU, 512 CUDA cores, 6GB memory (DRAM), 665 GFlops (double precision) (source: [Nvi])

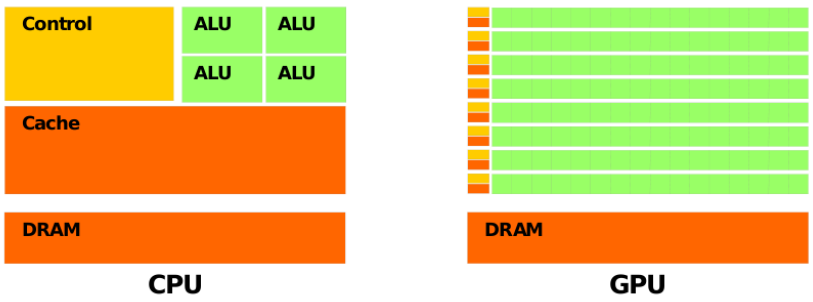


Figure 2.2.: Architecture of a quad core CPU (left) and a GPU (right). The GPU has hundreds of arithmetic logic units (ALU) which are grouped in multiprocessors (lines). (source: [Nvi13])

memory" that can make your code faster. Nevertheless by much easier means a very good program can be parallelized with CUDA. This is the aim of this introduction. The information in this chapter is based on the presentation on the Nvidia homepage [Nvi], a CUDA programming course at the HLRS in Stuttgart [HMWB13], from which I profited a lot, and my own experience.

The calculation speed of a perfectly parallelized program ($r_p = 1$ in Eq.1.1) grows linearly with the number of processors. Every real program performs less effective than this, although it is theoretically imaginable that a program could even run faster in parallel, because several data sets from one processor could be used on another one. This would happen only if the memory per processor is very limited. We assume that a higher parallelization is trivial in most cases but slows down the program on a limited number of processors. So we focus on the exploitation of the whole GPU computational power using the smallest possible parallelization (with optimal performance) that the hardware architecture allows. Fig. 2.2 shows a sketch of the design of a GPU and compares it to that of a CPU. The CPU consists of few arithmetic logical units (ALU) and comes with a bigger memory, see Tab. 2.1 for some exemplary numbers.

The GPU has several disadvantages. The first is the rigid architecture. Its computational power is comparable to a small cluster, but one must buy the

	CPU[Int]	GPU [Smi12]
product name	XEON E5 2670	Tesla K20x
cores	8	14 x 64 = 896
		(+14 x 192 = 2688 SP)
clock rate	2.6 GHz	732MHz
level 1 cache	8 x 32 kB	14 x 64 kB
		(+ 14 x 48 kB read only)
level 2 cache	8 x 256 kB	1536 kB
level 3 cache	20 MB	0
bandwidth	50 GB/s	240 GB/s
GFlops	200	1300
DRAM	64 GB to 750 GB	6 GB
power consumption	115 W	235 W
max devices per node	2	8

Table 2.1.: Comparison of the most important numbers for the latest CPU and GPU. (SP=single precision)

GPUs offered and can not configure them optimally to one's needs. For the aim of our project the limited RAM of 6 GB was the main drawback and the small cache limits the speed for many applications (We will go into detail later). To make it short, it has to be stated that CUDA threads are very small.

Notwithstanding, the obvious advantage is the pure computational power. With 1.3 TFlops the Nvidia K20x is unbeaten by any CPU or GPU. Another important number for fast computation is the bandwidth of the GPU RAM which is with 240 GB/s much faster than a CPU (50 GB/s) and the fastest DRAM (34 GB/s with dual channel). The main factor, which has so far not attracted much attention in physics, is the price. So here comes a cost calculation for the computation of the Arnold diffusion in the next chapter. Fig. 2.3 suggests that one optimally used GPU equals 100 CPUs. One cluster node with two M2090 GPUs costed 9000 € with a power consumption of 750 Watts which makes another 1300 € per year (assumingthe price 0.2 €/kWh). The standard CPU based cluster nodes costed 6000 € half a year later (November 2012), have 16 cores and consume 400 Watts (2 x 2.6 GHz 8-Core E5-2670

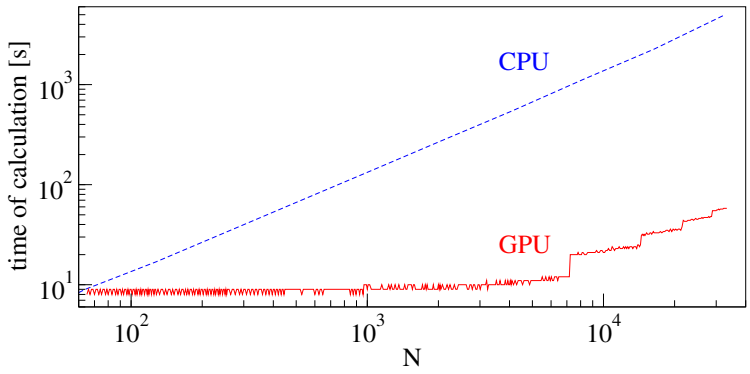


Figure 2.3.: Performance of a central processing unit (CPU) of the Augsburg University computational cluster (Intel Xeon Processor 2.93GHz Quad-Core X5570 Gainestown) versus the GPU (NVIDIA TESLA M2050) for a small dynamical system: The overall calculation time is depicted as a function of the number of realizations N [SDPH11]. For big particle numbers ($N > 7000$) the speedup is **two orders of magnitude** (in double precision).

Sandy Bridge). For the same simulation of Arnold diffusion that we performed on a cluster node with two GPUs we would have needed 200 CPU cores. Taking the air conditioning into account you can quite well say that the GPU needs **9 times less** in:

- acquisition costs
- power consumption
- real space

Even more important but less measurable is the gain in manpower. For a parallelization on CPUs (no matter how trivial it would be) one would need additional effort to handle the multiple computer nodes, write several files etc. With a GPU computer one doesn't even need a cluster. So even if it might look more complicated to write a CUDA program at first sight, once one gets used to it, it will be the easiest solution to parallelization.

With the CUDA language one can use almost all C features on GPU (device) code. The few things that can not be used inside device code are dynamically sized arrays, Input/Output management, System calls and long doubles. One

more software drawback is that the user can not execute common libraries on the GPU although the code might be written in C. But having a C source file is already the biggest step. With help of the CUDA libraries in the last section programs can be accelerated without knowledge of the CUDA language.

2.1. A first simple program

Here is the code for a simple program that uses the GPU. It calculates the Bessel function of the first kind $J_n(x)$ of orders $n = 0, ..., 255$ and arguments $x = 0, ..., 127$.

```
#include <stdio.h>
#include <cuda.h>

__global__ void calculate_Bessel_function (){
   double result = jn (blockIdx.x , double(threadIdx.x));
   printf ("\n jn( %d, %d) = %g", blockIdx.x , threadIdx.x, result);
}

main(){
   calculate_Bessel_function <<<256,128>>> ();
   cudaThreadExit();
}
```

In this code four new things appear to the C experienced programmer.

"`__global__`" tells the compiler that this function is called on the host (CPU) and executed on the device (GPU). Such a function is called a kernel.

"`<<<256,128>>>`" tells the compiler upon calling the kernel, how to distribute jobs on the multiple GPU cores.

"`threadIdx.x`" runs from 0 to the second number -1 in those brackets.

"`blockIdx.x`" runs from 0 to the first number -1 in those brackets.

So the total number of threads in this case is $256 \times 128 = 32768$. They are distributed over all GPU cores and every thread gets a different `threadIdx.x`

and `blockIdx.x`. To calculate Bessel functions on a CPU one needs an additional library. The speedup in comparison to the GNU scientific library (GSL) on a single CPU should be around 200. In addition the GSL libraries breaks down calculating Bessel functions of order 100.

Further CUDA built in variables that don't appear here are `blockDim.x` which is the second number in the brackets and `gridDim.x` which is the first. If the reader is wondering about the "`.x`" he should remember that GPUs were originally used for 3d graphics computation. The data type of all variables is `dim3` which has three members `x,y` and `z`. The members `y` and `z` are set to 1 if they are not specified. If they are specified like in

```
calculate_Bessel_function <<<256,(128,16)>>> ();
```

the corresponding variable `threadIdx.y` would run from 0 to 15 inside the kernel which would result in a 16 fold computation of the same value in our first simple program. Three or two dimensional indices are mapped onto a single index, where the x index changes faster than the y index which is again faster than the index.

The code inside a kernel is already device code. It is executed on a single ALU. If a kernel becomes bigger it should be divided into subfunctions. Functions that are called from device code must be prefixed with "`__device__`" on definition. With the prefix "`__host__`" or without any prefix a function can only be called and executed on the host. Functions that can be used from host and device must be prefixed with both "`__host__ __device__`".

2.2. Communication with the GPU

To make a useful program out of the fast one in the last section we need to transfer the data from the device to the host. Therefore we allocate memory on the host with

```
int memsize = 32768*sizeof(double);
double *bessel_values = (double*)malloc(memsize);
```

and memory on the device with

```
double *d_bessel_values;
cudaMalloc((void**)&d_bessel_values, memsize);
```

`cudaMalloc` allocates the memory on the DRAM of the GPU. Therefore the pointer to that data has to be changed and a double pointer needs to be passed. That `memsize` is the memory size in bytes like in the C function `malloc` is self explaining.

Next we must rewrite the kernel such that we can pass the pointer to `d_bessel_values` and copy the data to `bessel_values` on the host after the kernel execution with

```
cudaMemcpy(bessel_values, d_bessel_values, memsize,
   cudaMemcpyDeviceToHost);
```

It is again self explaining that one can copy data in two other ways with

```
cudaMemcpy(d_bessel_values, bessel_values, memsize,
   cudaMemcpyHostToDevice);
cudaMemcpy(d_bessel_values_2, d_bessel_values, memsize,
   cudaMemcpyDeviceToDevice);
```

for other purposes. Since CUDA 4.0 the address space of device and host is unified and the last argument can always be set to `cudaMemcpyDefault`. In host code data on the device can not be dereferenced. In device code data on the host can not be dereferenced except for data allocated with `cudaMallocHost` but this will be slow.

2.3. Parallelization using the GPU architecture

The GPU is divided into several multiprocessors. One block of a kernel launch is executed on a single multiprocessor which has 64 kB of memory shared among its 32 cores. Two numbers are important for data processing on the

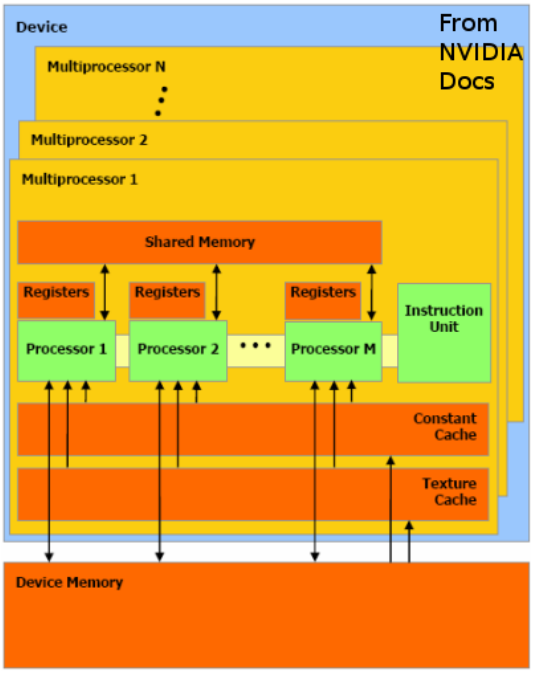


Figure 2.4.: Multiprocessor [Nvi13]: Several ALUs (32 for Fermi architecture) comprise a multiprocessor with shared memory. The whole GPU consists of several such multiprocessors.

device at this point. The bandwidth inside the global memory on the device is very high with 120 GB/s (M2090). On the new Kepler architecture this has even raised to 240 GB/s. The Bandwidths for comparable CPUs are 40 GB/s (X-5570) and 50 GB/s (E5-2670), the bandwidth of the latest DDR3 with dual channel is 34 GB/s per module. The bandwidth from host to device via PCIe is 6 GB/s. The distribution of threads via a kernel execution takes approximately $5\mu s$. In this time several thousands of cycles could have happened on one single ALU. So a first simple advice is: Don't talk to the GPU. Let it work. One should write algorithms that minimize data transfer from host to device, and - even better - write algorithms such that many instructions can be done on thread or multiprocessor level. It is sometimes useful to run code on the device even if it is slower than on CPU but the memory transfer via PCI can be avoided. And as many threads as possible should be run in parallel. So latency can be hidden with multiple warps per multiprocessor if shared memory suffices.

Threads of a block are executed in bunches of 32 such a group is called a warp. Therefore the `BlockDim.x` should be a multiple of 32. Also a warp is always executed on one multiprocessor. A program is more efficient if the threads of one warp use the same data since they share memory.

The device function `__syncthreads()` synchronizes the threads within the same block. The host function `cudaDeviceSynchronize()` synchronizes all threads of a kernel. Another way to synchronize threads, e.g. synchronizing just two blocks, is not given. If a complete synchronization is not needed but several threads write to the same memory like it is needed, when a histogram in global memory is filled with the position of particles calculated on the single threads the atomic functions are needed

```
int atomicAdd(int* address, int val);
```

If two threads add a number to the same address at the same time one of those operations will have no effect if a normal addition is used instead of the atomic. Atomic operations are 30% slower than their asynchronous counterpart if no threads interfere. If they interfere they are 200 times slower (on Fermi, on Kepler this has dropped to 60). So it should be used carefully not to destroy the whole performance gained. There are several atomic functions implemented but for writing an own atomic function the use of compare and swap `atomicCAS` is most convenient. So e.g. there is no `atomicAdd` function in double precision but Nvidia provides an example on [Nvi13].

2.4. Parallelization on multiple GPUs

One main advantage of graphic card computation is its compactness. Up to eight GPUs can be placed in one computer and easily utilized. The execution of a kernel on the GPU is synchronized only if the `cudaThreadSynchronize()` function is called or if data is transferred via `cudaMemcpy(...)`. This means the host code usually runs in parallel to the device code. So an easy way to use e.g. 2 GPUs on the same motherboard would be

```
cudaSetDevice(0);
calculate_Bessel_function<<<128,128>>>(d_bessel_values, ...);
cudaSetDevice(1);
calculate_Bessel_function<<<128,128>>>(d_bessel_values_2, ...);
```

Of course the memory handling becomes a bit more complicated since the programmer has to keep track of which data is on which GPU. The `cudaSetDevice` has to be called several times for memory allocation, data transfer and kernel execution. But for native parallelizable algorithms one whole order of magnitude can be gained by this technique.

2.5. Detailed memory model

The 20/80 Pareto principle suggests that 20% effort will suffice for 80% of the outcome. Those 20% have already been discussed, meaning they will suffice for 80% of parallel programs. For the final 20% of the cases some more numbers and details will be explored here.

Every multiprocessor has 64 kB memory that contains level 1 cache and shared memory. Shared memory is seen by all threads of a block. It is allocated inside a kernel via

```
__global__ void calculate_Bessel_function (){
   __shared__ double shared_values[128];
}
```

It is then treated like normal device memory. A powerful program copies data once from global to shared memory (500 cycles) and does a lot of calculations on that data (1 cycle) in one block without many synchronizations. The amount of shared memory is 16 kB by default and can be configured when calling a kernel by inserting a third number into the brackets like in

```
calculate_Bessel_function <<< 128, 128, 48>>> (...);
```

where 48 kB is already the maximum amount of shared memory and this memory is always taken from the 64 kB level 1 cache.

Another 64 kB of constant memory reside on the device. They are prefixed with `__constant__` in host code and since values can not be set directly on the device they have to be initialized with the function `cudaMemcpyToSymbol` or accessed (from host) via `cudaMemcpyFromSymbol`. This access is just useful for testing the code since no change of constant memory can be done on the device. Constant memory is visible for the whole device and cached with 8 kB cache per multiprocessor. This can save time. If one thread of a warp accesses one value in constant memory this value is cached on the multiprocessor and can be used several times again within one cycle by all other threads of that warp. The declaration is as follows

```
__constant__ double coordinates[8];
```

As mentioned before latency hiding can accelerate the program. E.g. while one thread requests data from global memory another thread can be executed on the same ALU. Therefore the number of threads per block should be a multiple of 32 (the warp size and the number of ALUs on Tesla) and the number of blocks per grid should be a multiple of the number of multiprocessors. The data in level 1 cache, shared memory, constant cache or texture cache can be read very fast from all ALUs of a multiprocessor. This can be exploited if one thread fetches the data from global memory (500 cycles) and all other threads perform calculations (of course not on that data) during that fetching time. Copying data from host to device does not only take a long time it also synchronizes the kernel. An alternative is the function `cudaMemcpyAsync(...)` that overlaps with a kernel execution. For this function the use of streams is necessary. Streams allow for parallel execution of kernels and data copying on a single GPU. They are useful if a lot of small kernels can be executed in parallel. The use is as follows

```
calculate_Bessel_function<<<128,128, shared_mem, 0>>>(d_bessel_values, ...);
calculate_Bessel_function<<<128,128, shared_mem, 1>>>(d_bessel_values, ...);
```

```
cudaMemcpy(bessel_values, d_bessel_values, memsize,
   cudaMemcpyDeviceToHost, 0);
cudaMemcpy(bessel_values, d_bessel_values, memsize,
   cudaMemcpyDeviceToHost, 1);
```

Kernels can be even launched from device on Kepler architecture. This is called dynamic parallelism. The nvcc compiler uses gcc for the compilation of host code (on Linux). If more efficiency is needed on the host code it is recommended to divide the source code into several objects, where one C/C++ object can be compiled with a faster compiler (like Intel's) and CUDA code are compiled with nvcc and both object files are linked afterwards.

2.6. Speedup through low precision

By using float precision a huge speedup can be gained on a GPU. Floats have 32 bits, one for the sign and 8 bits for the exponent, so the relative precision is $2^{-27} \approx 10^{-7}$. Doubles have 64 bits, where 11 are used for the exponent. Their relative precision is $2^{-42} \approx 2 \cdot 10^{-16}$. GPUs were originally designed for small short time mechanical equations that were only verified by the human eye so float precision was sufficient. For scientific purposes the newer GPUs are optimized for double precision. Still a factor of 2 is gained on Fermi (3 on Kepler) cards by using floats since there are more float precision ALUs on one multiprocessor. On the cheaper consumer cards the speedup is even higher. This is not the case of CPU programs since one multiply add of a double already takes the minimum of one cycle.

In addition to those single precision ALUs several special function units (SFU) reside in the multiprocessors. On Fermi there are 8 SFUs per multiprocessor. On Kepler there are 32. Such a SFU calculates the value of a transcendental function within one cycle. They are called via `__sinf`, `__expf` and so forth (cf. the manual [Nvi13]). Those SFUs can also perform single precision multiplications. The accuracy can be a little lower than that of the normal single valued functions like `sinf`. Therefore the argument should be chosen in the optimal interval which is $[-\pi, \pi]$ for trigonometric functions and $[0.5, 2]$ for

logarithms. To check whether the accuracy is sufficient one can write the code with the standard float functions `sinf` etc. and type "-use_fast_math" on compilation, so every `sinf` is treated as its fast counterpart `__sinf`. To improve accuracy and performance also the function `sinpi` can be advantageous to calculate values of the form $\sin(\pi x)$.

2.7. Libraries

There are several free CUDA libraries available for the most common functionality:

- CURAND
- CUBLAS
- CUFFT
- CUSPARSE

All of them need a little data for initialization and the routines can only be called in host code and have to be passed pointers to global device memory. The only exception is curand, which can generate random numbers in device code.

LAPACK routines used for more sophisticated linear algebra functions that are not covered by BLAS are not implemented by Nvidia but can be accessed via the libraries

- CULA
- MAGMA

They come with functions like diagonalization, singular value decomposition, Cholesky decomposition, QR-decomposition, least squares. They also provide BLAS functionality that often beats the cublas library. One member of the Magma developer team is Jack Dongarra, who helped developing the LINPACK benchmark for supercomputers, LAPACK, BLAS, MPI and many other libraries.

CURAND

The curand library produces high quality random numbers at high speed. So far it is unfortunately the only CUDA library to provide device functions which allow the programmer to use them inside kernels. The available distributions are

- uniform
- normal
- lognormal
- poisson

They are calculated with numerous recent pseudo and quasi random number generators (RNG) that can be chosen by the programmer. Quasi random numbers are correlated, which contradicts randomness, but they cover their probability space more uniformly and thus lead to faster convergence for some applications like Monte Carlo integration.

In the file "`curand.h`" the host functions are defined which generate samples of random numbers and store them in host memory. Needless to say that the bigger the samples, the more efficient is the execution. For the Mersenne Twister the sample size should be a multiple of $16384 = 2^{14}$. Before the first call the RNG has to be set up, which also takes some time and can be avoided in the further calls. Also like for any RNG a seed has to be set, which defines the initial conditions. The needed functions are as follows:

`curandCreateGenerator()`
`curandSetPseudoRandomGeneratorSeed()`
`cudaMalloc()` to allocate memory on device
`curandGenerate()` to generate numbers and copy them into host memory
`curandDestroyGenerator()` and `cudaFree()` to clean up

Instead of `curandGenerate()` which produces a random 32-bit unsigned integer the other functions like `curandGenerateUniform` or `curandGenerateNormalDouble` can be used to get the other distributions in single or double precision.

This random number generation on the host is very convenient if one wants to avoid an abundance of code. But of course it is more efficient to generate the random numbers directly in device code, the routines for which are in "`curand_kernel.h`". The two functions needed for this are

`curand_init()` to initialize the state of every single RNG in device code and

`curand()` to generate a random number

A nice feature of the device API is that it can produce discrete random numbers according to an arbitrary distribution. This distribution has to be passed as histogram. Both normal and lognormal distributions should be called via `curand_normal2_double` and `curand_log_normal2_double` since the underlying Box-Muller algorithm always produces 2 random numbers. There are even faster functions which generate four normally distributed random numbers in float precision or two in double precision relying on the Philox generator [SMDS11]. Initialize the random number generator with the function `curand_init`. Generate random numbers with `curand_uniform_double` and `curand_normal_double` and you get a random number with a period of 2^{190}. The generator is the XORWOW generator from [Mar03]. There are also other random number generators available.

Since generating random numbers does not involve a lot of data the speedup is expected to be high. We compare the `curand_normal2_double(...)` function to `nag_rngs_normal(...)` from the commercial NAG library. The GPU (M2090) produces as many random numbers as 150 CPUs (2.93GHz, X5570).

To use the curand library the header files `curand.h` and `curand_kernel.h` (for device functions) must be included and and the library must be linked with `-lcurand` on compilation.

CUBLAS

The CUBLAS library is an implementation of BLAS (basic linear algebra subprograms). It performs multiplications and additions of all kinds like

$$C = \alpha AB + \beta C \tag{2.1}$$

where A, B, C can be vectors or matrices. It has efficient routines for general cases, triangular, banded and/or symmetric/hermitian matrices. In addition to the normal BLAS routines it can do a LU decomposition. It can solve a system of linear equations and therefore calculate the inverse of a matrix. And it has so called "batched" functions if the same operation has to be done on many smaller system, which is especially advantageous for the GPU.

Matrix multiplication is often used as a first example of an easy CUDA program. This is a very bad choice, since its naive implementation has numerical effort $\mathcal{O}(N^3)$ whereas an intermediate implementation, the Strassen algorithm [Str69], described in [PTVF02], is $\mathcal{O}(N^{\log_2 7 \approx 2.8})$ and the best currently available Coppersmith-Winogard algorithm [CW90] is even $\mathcal{O}(N^{2.37})$, which makes a huge difference (≈ 1000) in the case of big matrices like those of a quantum system with 40.000 states. Nevertheless the way more advanced implementation pays off only for much bigger matrices and the implemented algorithm in most libraries still is the Strassen algorithm.

So far the cublas library can hardly outperform standard CPU libraries like Intel's MKL. The matrix size must be bigger than 1000 to keep all CUDA cores busy. On the other hand a matrix of size 10000 almost fills the DRAM. Within this region matrix multiplication with cublas can be as fast as the MKL on 32 XEON cores. This is good but not brilliant. But it has to be stated that the handling of big matrices is quite contrary to the GPU architecture. And since CUDA is a new field improvements can be expected.

The usage is easy to the BLAS experienced programmer. After including the header file "`cublas.h`" and an initialization with

```
cublasHandle_t handle;
cublasCreate (&handle);
```

the functions can be called in the standard BLAS nomenclature with an additional reference to the handle variable like

```
cublasZgemm (handle, ...);
```

for complex matrix-matrix multiplication, where the dots "..." stand for the arguments used in all other BLAS implementations. Upon compilation the library must be included with `-lcublas`.

Further routines of the BLAS library are Givens rotations, rank one updates for vectors and matrices X, Y

$$A = A + \alpha X Y^{\dagger} \tag{2.2}$$

scaling, multiplication addition, scalar product, norms, swapping (e.g. for sorting a basis). A function especially useful for quantum mechanics is `cublasZgemm` which computes

$$C = \alpha Op(A) + \beta Op(B) \tag{2.3}$$

where $Op()$ can be the identity, the transpose or the adjoint (or conjugate transpose in the manual) of a matrix and `cublasZdgmm` which multiplies a diagonal matrix stored in vector format with another matrix from left or right.

CUFFT

With the CUFFT library the programmer can perform one to three dimensional fast Fourier transforms from real to complex, complex to complex or complex to real (if symmetric) according to

$$X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i k n}. \tag{2.4}$$

The transform size N should have prime factors 2, 3, 5 and 7 only to profit from the Cooley-Tukey algorithm [CT65] with effort $\mathcal{O}(N \log N)$. The Cooley-Tukey algorithm was originally invented by Gauss around 1800 but of course

at those days no one considered computational effort for large numbers. So the idea wasn't even broadcast by Gauss [HJB85]. If the size of the Fourier transform is not a product of those numbers the slower Bluestein algorithm [Blu70] is employed. For its usage the "`cufft.h`" header file has to be included in the source code and the library has to be linked with `-lcufft`.

CUSPARSE

Linear algebra becomes tough if you go to big problem sizes. A complex matrix of size 50.000 already occupies 40 GB in memory and their handling takes a huge time (e.g. diagonalization takes approximately one day growing with $\mathcal{O}(N^3)$). For realistic problems that pass this border it is very rare that in a matrix all values are nonzero or even of same size.

A nice physical example is a spin chain of length N with Hamiltonian

$$H_{XYZ} = \sum_j J_X S_j^x S_{j+1}^x + J_Y S_j^y S_{j+1}^y + J_Z S_j^z S_{j+1}^z + 2h \sum_j S_j^z. \tag{2.5}$$

Its Hilbert space $\mathcal{H}$ has dimension $dim_{\mathcal{H}} = 2^N$ so the full Hamiltonian would occupy $16 \cdot 2^{2N}$ bytes in memory and the limit would be around 15 spins. But the entries are much less than this. A single bond Hamiltonian has the form

$$h_{j,j+1} = \begin{pmatrix} \frac{1}{4}J_z + 2h & & & \frac{1}{4}(J_x - J_y) \\ & -\frac{1}{4}J_z & \frac{1}{4}(J_x + J_y) & \\ & \frac{1}{4}(J_x + J_y) & -\frac{1}{4}J_z & \\ \frac{1}{4}(J_x - J_y) & & & \frac{1}{4}J_z - 2h \end{pmatrix} \tag{2.6}$$

when acting on the corresponding bond only. It has 8 entries. When acting on the whole system it is the tensor product

$$H_{j,j+1} = h_{j,j+1} \otimes \mathbb{1}_{N-2} \tag{2.7}$$

with an identity for the Hilbert space of the remaining spins. So the whole matrix has $8 \cdot 2^{N-2}$ entries and the whole Hamiltonian as a sum of N such

bonds has $N \cdot 8 \cdot 2^{N-2} = N2^{N+1}$ entries, which fills the memory around 30 spins, or the other way around it occupies only 16 MB for a spin chain of 15 spins.

If a matrix consists of zeros mainly it is called sparse. If this is the case a big amount of memory and time can be saved if all the zeros (or almost zeros) are avoided in storage and calculation. Banded matrices can already be done with cublas. For other cases of sparsity the CUSPARSE library provides basic linear algebra operations like

- multiplication / addition
- Givens rotations
- solution of linear equations
- conversion of sparse formats

Compared to an Intel Xeon x5680 hexacore with the MKL library the performance ranges from "as fast" to 40 times faster depending on the application and problem size according to [Nvi].

There s just one choice for the format of a sparse vector. It is always stored in two arrays one of which contains the values of all nonzero elements and the other array contains the position of those elements. For matrices several options appear. A dense matrix is generally stored in column major format meaning it is a vector in memory in which the columns of a matrix are stored one after another. This is the same as for cublas.

The sparse format of a matrix could be taken as the sparse vector format of this corresponding vector in memory, but this is disadvantageous for computation. CUSPARSE offers a lot of formats for the most frequent types of sparsity. All of them contain 3 arrays. The easiest is the coordinate format (COO) in which the nth entry of the two integer arrays contains the line and column of the nonzero entry on nth position.

$$\begin{pmatrix} & & 3.7 & & \\ 2.9 & & & 7.7 & \\ & 7.1 & & & \\ & 7.4 & & 0.9 & 0.8 \\ & & & & 1.2 \end{pmatrix} \xrightarrow{\text{COO}} \begin{matrix} \begin{pmatrix} 3.7 & 2.9 & 7.7 & 7.1 & 7.4 & 0.9 & 0.8 & 1.2 \end{pmatrix} \\ \begin{pmatrix} 0 & 1 & 1 & 2 & 3 & 3 & 3 & 4 \end{pmatrix} \\ \begin{pmatrix} 2 & 0 & 3 & 1 & 1 & 3 & 4 & 4 \end{pmatrix} \end{matrix} \tag{2.8}$$

So the nonzero elements are stored line after line which corresponds to a row major format. Since the rows are sorted in ascending order and most row numbers occur many times it suggest itself to shorten the vector containing the row which is done in the Compressed Sparse Row format (CSR)

$$\begin{pmatrix} & & 3.7 & & \\ 2.9 & & & 7.7 & \\ & 7.1 & & & \\ & 7.4 & & 0.9 & 0.8 \\ & & & & 1.2 \end{pmatrix} \xrightarrow{\text{CSR}} \begin{matrix} \begin{pmatrix} 3.7 & 2.9 & 7.7 & 7.1 & 7.4 & 0.9 & 0.8 & 1.2 \end{pmatrix} \\ \begin{pmatrix} 0 & 1 & 3 & 4 & 7 & 8 \end{pmatrix} \\ \begin{pmatrix} 2 & 0 & 3 & 1 & 1 & 3 & 4 & 4 \end{pmatrix} \end{matrix} \tag{2.9}$$

where the column vector is the same as in COO format and the row vector tells which entry in the dense value vector is the first in the next line of the original matrix (and the last entry is the number of elements plus the first number). Similar is the Compressed Sparse Column format (CSC) which is CSR format with switched role of lines and columns. So for this small matrix with 25 entries the memory space could be reduced from 200 bytes to 128 bytes in COO and 120 bytes in CSR. For the several other formats for frequent types of sparse matrices I refer to the manual [Nvi13]. The CUSPARSE library is used by including the header file "`cusparse.h`" and linking by `-lcusparse`.

3. Arnold diffusion

This chapter is devoted to the Arnold diffusion (AD) which is named after the Russian mathematician Vladimir Arnold [Arn64]. AD does not need many particles to appear. In fact, it appears whenever the number of degrees of freedom (DOF) is $M \geq 3$. AD plays a role in celestial mechanics [Con02, Mor02], plasma dynamics and tokamaks [Lic92, WP01], the evolution of Rydberg atoms in crossed magnetic fields [vDU96], and it may explain experimentally observed effects of emittance growth in the TeVatron colliders [OO07] and even resolve the fundamental issue of the ultraviolet cut-off in statistical mechanics [BGG84].

The history of Hamiltonian dynamics is vivid. In the beginning of the 20th century it was believed that a Hamiltonian system with many DOFs were always ergodic, i.e. every open set in phase space is connected. This was even "proved" by Fermi in 1923 [Fer23]. Nevertheless in the 1950s the Kolmogorov-Arnold-Moser theorem (KAM theorem) was proved [Kol54, Mos62, Arn63]. It stated, that the main part of the phase space consists of invariant manifolds, so called KAM tori, and chaos is a rather marginal thing. So the picture was upside down, and Fermi's reasoning was found to be wrong. Then in 1964 Arnold changed the situation again by proving that in the system

$$H(x, y, p_x, p_y, t) = \frac{p_x^2}{2} + \frac{p_y^2}{2} + \epsilon \left[\cos(x) - 1\right] \left\{1 + \mu \left[\sin(y) + \cos(t)\right]\right\}, \quad (3.1)$$

which is a prototype of any weakly perturbed nonlinear Hamiltonian with three DOFs, exists a trajectory initially arbitrarily close to the origin which leads to a point arbitrarily far from the origin, no matter how small the perturbation parameters ϵ, μ are.

This phenomenon was thereafter called Arnold diffusion. Thus the picture

of a Hamiltonian phase space with $M \geq 3$ DOFs is very motley. There are fully chaotic regions, chaotic regions with stable KAM tori which have zero measure and regions, where KAM tori fill the phase space almost up to full measure and chaotic channels are a minority. Arnold's paper concerned the latter situation. But the term AD is nowadays mainly used for a diffusion along a web of resonances which occurs in the intermediate regime [LL92, FGL00].

To investigate this phenomenon this chapter is organized as follows. First an introduction to Hamiltonian chaos on the route from one DOF to many DOFs is given. Then the computational algorithm used is introduced. The basic concepts of AD are stated in the third section. Subsequently the AD is probed with a ratchet setup. The Nekhoroshev estimates for the diffusion speed are reviewed with an experimentally feasible setup. Finally, a brief comparison to the normal usage of the term diffusion is made and diffusion coefficients are calculated.

3.1. Classical chaos in Hamiltonian systems: From one to three degrees of freedom

The AD is the transition from chaos to integrability in some sense. So we have two choices for the introduction. We can start from one DOF, where we always have integrability and then move towards many DOFs, or we can proceed from an integrable system, which can have a lot of DOFs that are not coupled, to a non-integrable one, by coupling them.

The phase space Ω of a Hamiltonian system with one DOF $M = 1$ is two dimensional, because the DOF, e.g. space coordinate x, comes with a conjugate variable, e.g. momentum p. Since the Hamiltonian or the energy $E = H(x, p)$ is an integral of motion, any such system is integrable. The prototype of an integrable Hamiltonian is one that depends on action coordinates only

$$H_0(\mathbf{q}, \mathbf{p}) = H(\mathbf{p}), \tag{3.2}$$

for which every solution has a constant action $\mathbf{p}(t) = \mathbf{p}_0$ and linearly growing

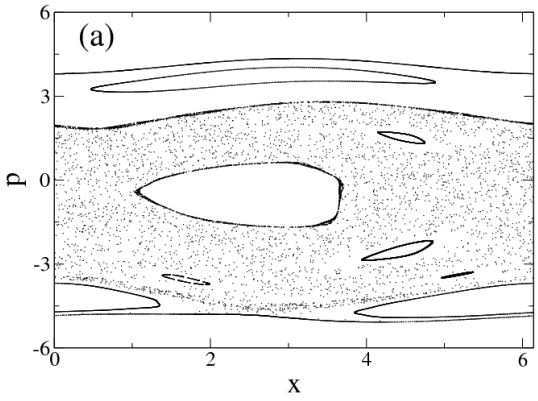


Figure 3.1.: Poincare section of the one dimensional nonlinear driven system Eq. 3.5, with Hamiltonian $H = \frac{p^2}{2} + \cos(x) + xE(t)$ and driving $E(t) = 2\cos(2t) + 2\sin(4t)$ (Adapted from [DMMFH07]).

angle $\mathbf{q}(t) = \mathbf{q} + \omega t$ where $\omega = \nabla_p H(\mathbf{p}_0)$ according to Hamilton's equations of motion.

If one adds another DOF so that $M = 2$, one has two choices, which turn out to be alike. One could keep the system autonomous with another space variable $H(x, y, p_x, p_y)$ or one could add an explicit time dependence $H(x, p_x, t)$. In the first case the Hamiltonian remains an integral of motion and every trajectory is restricted to the three-dimensional shell of constant energy in the overall four-dimensional phase space $\Omega_{\text{aut}} = \{(x, y, p_x, p_y)\}$. The time dependent system has no such integral of motion but its phase space $\Omega_{\text{driven}} = \{(x, p_x, t)\}$ is three-dimensional. The typical Hamiltonian of the form

$$H(x, p_x, t) = \frac{p_x^2}{2} + V(x, t) \tag{3.3}$$

which has no products of p and q can be canonically transformed into an autonomous system with one additional DOF substituting the time. By treating the time as a spatial coordinate τ with derivative $\dot{\tau} = 1$ we see that the corresponding momentum p_τ must occur linearly in the Hamiltonian, so that the Hamiltonian equation $\dot{\tau} = \partial_{p_\tau} H = 1$ is fulfilled. Thus, it is independent of this new momentum p_τ which is governed by $\dot{p}_\tau = -\partial_\tau V(x, y, \tau)$. This latter differential equation is not needed for the dynamics but can be used for checking

the numerical error of the so obtained autonomous Hamiltonian

$$H(x, \tau, p_x, p_\tau) = \frac{p_x^2}{2} + p_\tau + V(x, \tau). \tag{3.4}$$

For brevity only the autonomous case will be considered in the following introduction, although we will use mainly driven systems. On its three dimensional manifold the trajectory can exhibit chaotic behavior, which is most easily defined as exponential growth of a deviation between two initially close points. Nevertheless there can be barriers which prevent the particle from exploring all accessible phase space. The KAM theorem predicts the existance of stable KAM tori in phase space regions that are close to integrable. The dimension of these tori equals the number of the system's DOFs. They form a torus after a proper canonical transformation to action angle variables. A trajectory initiated on such a torus will cover its whole surface in quasiperiodic manner. According to the uniqueness theorem by Picard-Lindelöf, such a trajectory can not be intersected by others and therefore constitutes a border to chaotic regions. Moving to higher momentum these KAM tori form a dense Cantor set in phase space with nonzero Lebesgue measure. Nevertheless no KAM torus has inner points. Thus one can not distinguish numerically between trajectories lying on such a torus or between two of them. In Fig. 3.1 the Poincare section of a driven pendulum

$$H_{1D}(x, p_x, t) = \frac{p_x^2}{2} + \cos(x + E^x(t)); \quad E^x(t+T) = E^x(t) \tag{3.5}$$

is shown. In this figure we see that a chaotic layer has emerged around the separatrix of the undriven pendulum and inside this layer some stable islands are submerged, which encircle resonant points. Adding the next DOF $M = 3$ a crucial thing changes. KAM tori still exist, even a lot of them. But their dimension equals the number of DOFs, whereas the dimension of the phase space has grown to five. As one can imagine easily, a three dimensional manifold can not provide a separation of a five dimensional space. Thus, a chaotic manifold can cover the whole phase space. On the other hand the whole set of invariant KAM-tori Ω_{tori} has full measure and could provide a separation. As an obvious first example, we use two coupled driven pendula

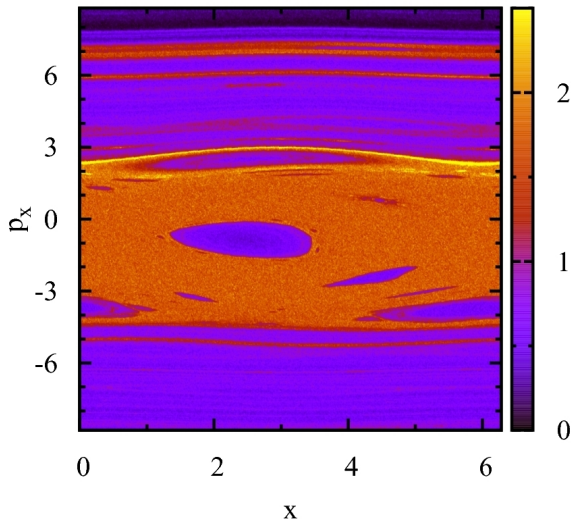


Figure 3.2.: Density of Poincare sections $\bar{W}(x, p_x, t = 10^8)$ of the first driven pendulum of Eq. 3.6, obtained with a single trajectory initialized at $x = 0, p = 0$. The coupling strength is $\epsilon = 0.1$. Darker areas coincide with the stable islands from the uncoupled system in Fig. 3.1. Several barrier tori are visible. There are denser lines at integer momenta $p_x = -4, 5, 6$.

$$H_{2D}(x, y, p_x, p_y, t) = H_{1D}(x, p_x, t) + H_{1D}(y, p_y, t) + \epsilon \cos(x - y). \tag{3.6}$$

To keep track of this system, the question is what to plot on this two dimensional sheet. A first guess is, since we are dealing with a weakly coupled system, to plot just the phase space of the first pendulum, i.e. the $\{x, p_x\}$ manifold. This is presented with Fig. 3.2. Since a single trajectory penetrates a big part of the phase space, we would just see a black area by drawing the positions. Therefore we add a color bar, that tells us the frequency of the particles visits at that point in phase space, and we get the distribution

$$\bar{W}(x, p_x, t) = \sum_{n=0}^{[t/T]} W(x, p_x, nT). \tag{3.7}$$

Luckily, due to the relatively weak coupling, $\epsilon = 0.1$, the chaotic islands from

Fig. 3.1 have not changed at all, at least at this scale. So instead of searching the whole space for resonances in a system with two DOFs, one can couple the system weakly to another DOF and let a single trajectory search for those resonances.

At this point it is appropriate to discuss a necessary quantity for later use, the winding number ω_x. The big stable islands seen in Figs. 3.1 and 3.2 are the surrounding of a major resonance.The biggest island in the center has average momentum $\bar{p}_x = 0$. Inside this island are all stable trajectories of the librations of the system, which stay within one well of the potential. The next bigger island below has average momentum $\bar{p}_x = -2$, which means that the trajectory inside moves exactly one spatial period to the left during one period of the driving. So the winding number of this trajectory is $\omega_x = -1$. The next bigger island at the lower border of the chaotic sea has average momentum $\bar{p}_x = -4$. This is also the average momentum of the resonant trajectory in its center with winding number $\omega_x = -2$. In between are two smaller islands at momentum $p_x = -3$. They actually belong together and have a non-integer winding number $\omega_x = -\frac{3}{2}$. Generally for every resonance an integer vector $\mathbf{k} \in \mathbb{Z}^n$ can be given so that

$$\mathbf{k} \cdot \omega = 0 \tag{3.8}$$

where $\omega = (\omega_x, \omega_y, \omega_t)$ is the vector of the winding numbers of that resonance. A resonance can always be written as an integer vector with the entries giving the ratios of the particular winding numbers of the DOF. So e.g. if we write $\omega = (\omega_x, \omega_t) = (-3, 2)$ we mean that the systems moves -3 periods in x direction while it moves 2 periods in t direction.

Next, the evolution in momentum and the extended real space is of interest. For this we propagate a whole ensemble of particles from the origin in momentum subspace $\mathbf{p} = 0$ and watch their evolution in the momentum plane and the space plane. Fig. 3.3 shows the output of a single day simulation of system Eq. 3.6 for approximately 10^6particles and a final time of $t_f = 10^6 T$ and 200 steps per period. Initially the particles where spread uniformly over the central unit cell of the periodic potential $(x_0, y_0) \sim U(0, 2\pi)^2$ and had almost zero velocity. According to Arnold, the particles ought to diffuse along resonance

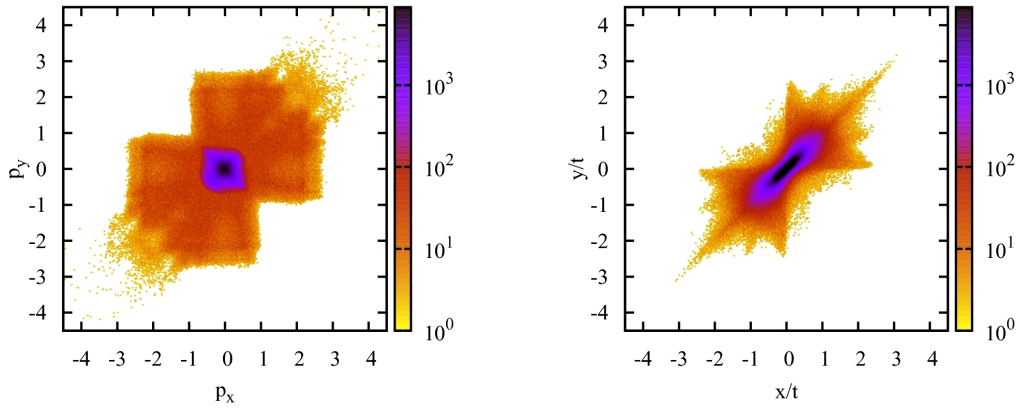


Figure 3.3.: Probability density of momentum $W(\mathbf{p}, 10^5 T)$ and propagator $W(\mathbf{r}, 10^5 T)$ (not normalized) of an ensemble of 10^6 particles initially at $x = 0, p = 0$.

channels. Some borders are visible and also some lines which could represent resonances. But there is no clear picture of a resonance web in phase space. To keep track of the movement in between initial and final time, we exploit the stickiness of the resonances. Resonance lines are surrounded by hardly penetrable tubes formed by KAM-tori and other fractal structures (cantori), which causes a particle to stay inside such a tube once it has entered. So we compute the average velocity $\mathbf{v}$ of each particle over a time interval of several driving periods $\Delta t = nT$

$$\mathbf{v}_{\Delta t}(t) := \frac{\mathbf{r}(t) - \mathbf{r}(t - \Delta t)}{\Delta t}. \tag{3.9}$$

Generally we would need the action variables that bring the Hamiltonian closest to an integrable form. But this canonical transformation depends on the position in phase space and has no chance to be done analytically. Calculating the average velocity seems to be a good midway, that is numerically feasible. Fig. 3.4 shows the histogram of the average velocities. For the sake of more statistical data, we do not only compute the histogram $W(\mathbf{v}_{100T}, 10^6 T)$ over the last time span Δt, but the histograms after every Δt are added up to an

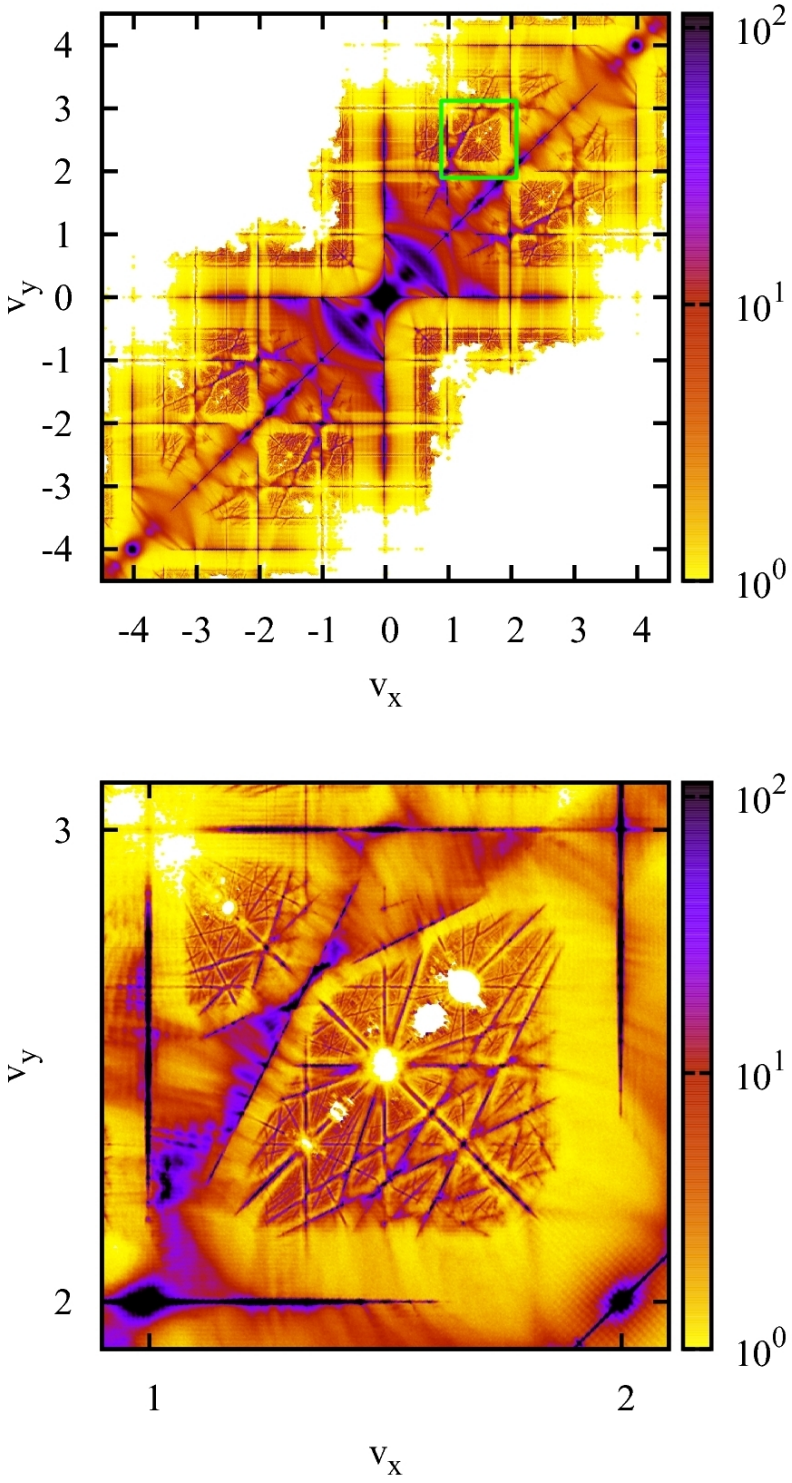
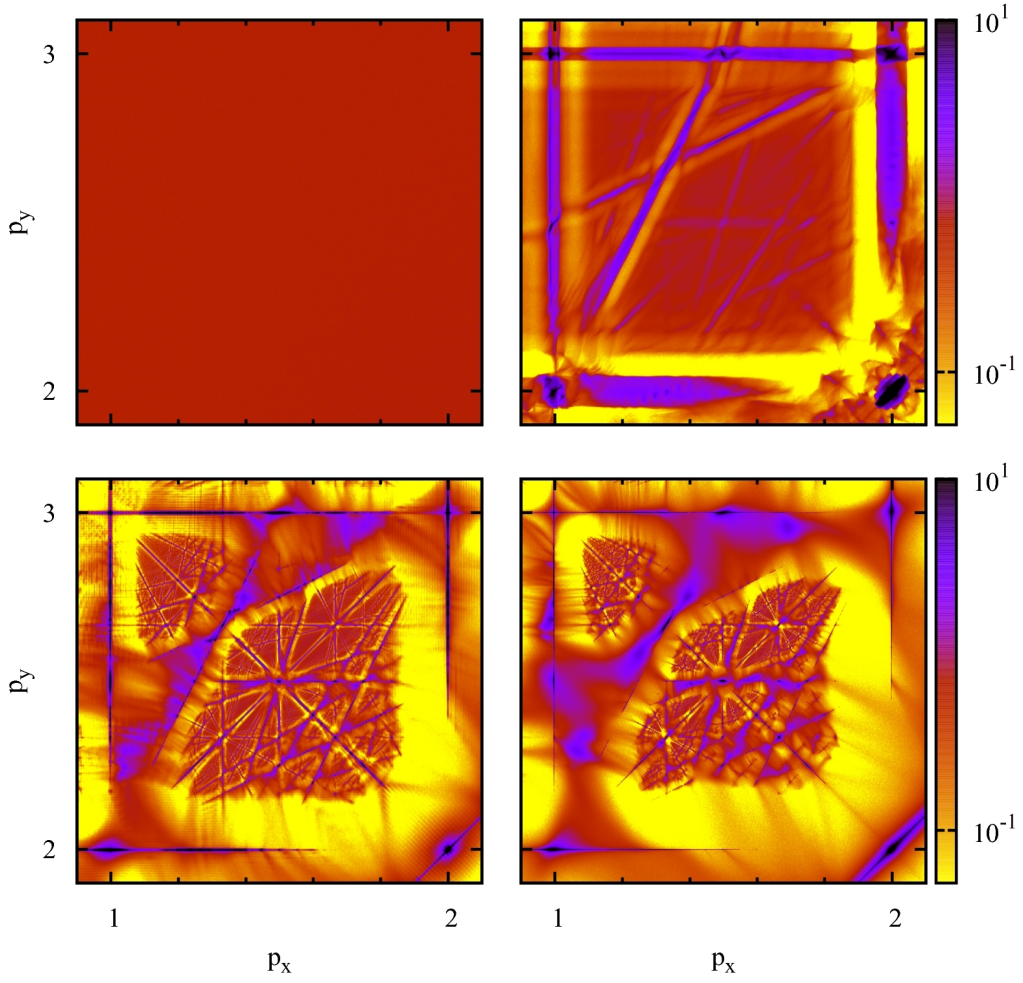


Figure 3.4.: Probability density of average speed $\bar{W}(\mathbf{v}_{100T}, 10^6 T)$ of system 3.6 with $\epsilon = 0.1$ (not normalized). The ensemble was initially at rest. The zoom shows holes of the distribution around double resonances produced by a dense set of KAM tori.

a) $\Delta t = T$ b) $\Delta t = 16T$



c) $\Delta t = 128T$ d) $\Delta t = 1024T$

Figure 3.5.: Histograms for different averaging times Δt. The resolution of the resonance channels improves with growing Δt, but "leaky" regions are lost.

average

$$\bar{W}(\mathbf{v}_{\Delta t}, N\Delta t) = \frac{1}{N}\sum_{i=1}^{N} W(\mathbf{v}_{\Delta t}, i\Delta t). \tag{3.10}$$

In this representation vertical lines are formed by resonances of x and t, horizontal lines by y,t-resonances and diagonal lines by x,y-resonances. At the main lines the ratios of the resonant winding numbers are integer. There are also resonances in the smooth parts, where no lines are seen. These resonances are librations and have therefore average speed zero.

A histogram of the average velocity is useful to resolve the diffusion on the Arnold web (AW). The averaging time Δt was set arbitrarily and is not determined by the system so far. What happens if this parameter is changed is depicted in Fig. 3.5. Starting with a uniform distribution in momentum space, the average speed in the interval stays uniform at least in the chosen area and within the 2 orders of magnitude color scale. With $\Delta t = 16T$ a lot of resonances can bee seen. Further increment of Δt improves the width of a resonance. If a particle is caught inside a channel of width Δv_r at a resonance with velocity $\mathbf{v}_r$ the width of the velocity distribution decreases with $1/\sqrt{\Delta t}$. But this is only true if the particle sticks to that resonance over the whole averaging interval. If it does not, the resonance is not resolved at all and for growing Δt "leaky" parts of the resonance channels are lost. So the appearance of the result is a question of timescales. As in Fig. 3.4, we choose the averaging time $\Delta t = 100T$ in the following for simple technical reasons. Lower values do not reveal much of a structure. For higher values we get less statistical data. Nevertheless this parameter is arbitrary and its choice will change the output.

3.2. Computational methods and algorithms

The numerical recipe is the key ingredient since AD is about exponentially small perturbations and exponentially large times. The common value to determine chaoticity is the Lyapunov exponent. The standard way for a representation of the AW are finite-time Lyapunov exponents [FGL00, MPV86, TK07] or the frequency analysis [DL93]. The average speed is used here, since it reflects the time evolution of the whole diffusion process and needs no further

data processing.

The Hamiltonian flow of our setup is symplectic, i.e. it conserves phase space volume. A numerical scheme which makes use of this fact is the symplectic integrator. It computes the time evolution of a slightly perturbed Hamiltonian, but it is symplectic up to machine precision. Since AD demands a very long integration time t_f compared to the systems characteristic time $T \ll t_f$, the standard Runge-Kutta algorithm would produce a growing error. In this work the symplectic integrator of fourth order by Forest and Ruth [FR90] was used, as well as the sixth order integrator by Yoshida [Yos90]. The SABA integrator [LR01] for small perturbations was found to be optimally suited in the case of the slowest diffusion. A short recipe for the integrator is given in the appendix. The timescale of all systems is given by the period T_x of the oscillations at the bottom of the potential wells or by the period T_t of the driving which for tangible AD to occur should be close to T_x anyway. Since the particles reside in regions of the phase space with very different oscillation frequencies, a low order integrator with a small time step has to be preferred to a high order integrator with big time step. This is especially true for the driven case where the system is consequently heated up and accelerated.

Some remarks on the numerical efficiency must be given here. The Calculations to Fig. 3.7 took one day and involved 32000 particles up to a time of $t_{final} = 10^7 T$ where every period T was divided into 200 time steps $dt = T/200$ this makes a total of $\approx 10^{14}$ time steps. To evaluate the efficiency, Nvidia's manufacturer specifications are taken into account, which say the M2090 GPU has 0.6 GFlops ($5 \cdot 10^{16}$Flop/day). This means, our algorithm needs approximately 500 Flop per time step on average. A fourth order method was used which divides one time step into another 4 equal steps and finally for every spatial dimension a sin and a cosine must be evaluated, which is the most time consuming part of the ODE. This leaves approximately 30 Flop per sin evaluation[1]. Mind that this includes all necessary multiplications, additions, read and write operations. So there's not much room for numerical improvement to the implementation. The bottleneck are the trigonometric functions. By

[1] With the function sincos() a sin and cos can be evaluated with 1,5 times the effort of a sin only

lowering accuracy an improvement can be made. This is reasonable if one is just interested in the AD itself and does not care so much about the specific system. In this case the number of time steps can be reduced even to one for low perturbation strength as well as float precision suffices for many parts of the computation. [2]

3.3. Introduction to the Arnold diffusion

In Sec. 3.1 we have discussed three-dimensional Hamiltonian chaos and we have already seen an AW. But for a proper introduction to AD we must start from the integrable case. An integrable Hamiltonian $H(\mathbf{p}, \mathbf{q})$ can be transformed to action angle coordinates $(\mathbf{J}, \phi)$ so that it depends on actions only

$$H(\mathbf{J}, \phi) = H(\mathbf{J}). \tag{3.11}$$

thus the action variables are constant $\dot{\mathbf{J}} = -\nabla_\phi H = 0$ and the angles grow linear in time $\phi(t) = \phi_0 + t\nabla_J H$. The frequency or winding number in one direction is $\omega_i := \partial_{J_i} H$. The Hamiltonian is assumed to be periodic in the angles with period 2π. We shall restrict our analysis to the three dimensional case here, which is sufficient for AD and best for our imagination. A particle in its 3 dimensional angle space $[0, 2\pi)^3$ will move into some direction and reenter this cube on one side, once it has left it on the other side. In this way it will intersect the cube on and on. If the trajectory hits itself, the time evolution can be stopped, since one knows it will just do the same again. This can only happen, if all pairs of winding numbers have a rational ratio, or in other words are linearly dependent over $\mathbb{Z}$, i.e.

$$\exists \mathbf{k} \in \mathbb{Z}^3 / \{0\} : \mathbf{k} \cdot \omega = 0. \tag{3.12}$$

For this case all DOF are in resonance. If the winding numbers are linearly independent over $\mathbb{Q}$ the trajectory can never hit itself and will therefore fill the whole cube $[0, 2\pi)^3$. Such a trajectory or rather the volume it covers is called a

[2] The fast math function ___sinf() will possess the same periodicity as long as the argument is small which is the only prerequisite needed for AD.

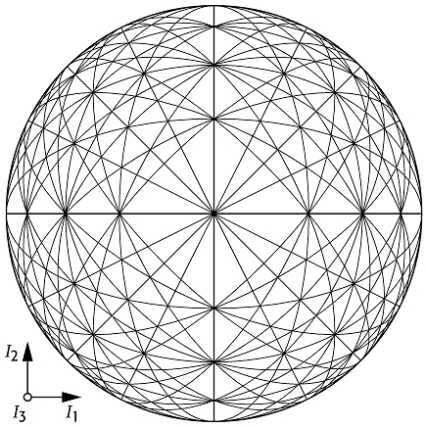


Figure 3.6.: Sketch of a resonance web. Some resonance lines are drawn on the shell of constant energy of the action variables I_i along which particles diffuse upon perturbation. (Adapted from [TLL79])

KAM torus. The single case left is that only two winding numbers are linearly dependent and one is independent. Such a trajectory will cover several planes in the cube. This is a single resonance and a 2 dimensional KAM torus. The type of trajectory is therefore determined by the initial actions $\mathbf{J}$ only. And the whole cube for a certain J is either covered by a two dimensional set of initial conditions for the double resonance, by a one dimensional set in the case of a single resonance and a single trajectory in the case of the full KAM torus.

Chaos emerges as soon as a perturbation of this Hamiltonian is introduced

$$H(\mathbf{J}, \phi) = H_0(\mathbf{J}) + \epsilon H_1(\mathbf{J}, \phi). \tag{3.13}$$

No matter how small the strength of the perturbation ϵ is all the resonances which were independent of the initial angle $\phi(0)$ are mixed. Just a single resonance will survive. Since the single resonance has no measure we will never be able to calculate its trajectory. Also the KAM tori around a resonance will be destroyed and mixed into a chaotic sea. At this point the KAM theorem comes into play. It states, that if ϵ is small, there are constants $\gamma > 0, \tau > 0$ for the system so that every KAM torus survives that is distant enough from

all resonances in the sense, that for its winding number vector ω the inequality

$$|\mathbf{k} \cdot \omega| \geq \frac{\gamma}{|\mathbf{k}|^\tau} \forall \mathbf{k} \in \mathbb{Z}^n \tag{3.14}$$

holds. So every resonance cuts out a chaotic channel in the phase space. All together the resonances form a web which has a measure of $\mathcal{O}(\gamma)$. An illustration of this web on a spherical energy shell is depicted in Fig. 3.6. For a numerical investigation of this diffusion mechanism we use an autonomous Hamiltonian

$$H(\mathbf{P}, \mathbf{X}) = \frac{\mathbf{P}^2}{2} + \varepsilon H_p(\mathbf{X}) \tag{3.15}$$

with coordinates $\mathbf{P} = (p_x, p_y, p_z)$ and $\mathbf{X} = (x, y, z)$. As perturbing nonlinear potential we take

$$H_p(\mathbf{X}) = \cos(x)\cos(y)[1 + \cos(2z)]. \tag{3.16}$$

It should be noted that the simpler potential $\cos(x)\cos(y)\cos(z)$ would not lead to AD, since the additional symmetry bears an integral of motion that reduces the DOFs to two. It is actually not necessary to keep the perturbation strength as a variable since the strength ε_{eff} is given by the ratio of the potential energy to the total energy

$$\varepsilon_{\text{eff}} = \max |H_p(\mathbf{X})| / H(\mathbf{P}, \mathbf{X}) = 2/E. \tag{3.17}$$

We start an ensemble of particles at a point of strong chaos, the double resonance $p_x = p_y$, $p_z = 0$. The initial space coordinates are uniformly distributed over the interval $[0, 2\pi]$ and the initial momenta assume a Gaussian distribution of small width $\sigma_p = 0.01$. All three momenta are then rescaled to ensure the exact initial energy of $E = 15$. The time evolution is shown in Fig. 3.7.The ensemble stays close to the initial resonant point until $t = 10^3 T$. Then the fast (so called Chirikov) diffusion into the chaotic sea around it happens, covering one sixth of the sphere after $t = 10^4 T$. Consequently, the ensemble spreads into the symmetric parts after the next order of magnitude. Until the end of the calculation the points where the energy is concentrated in one single DOF,

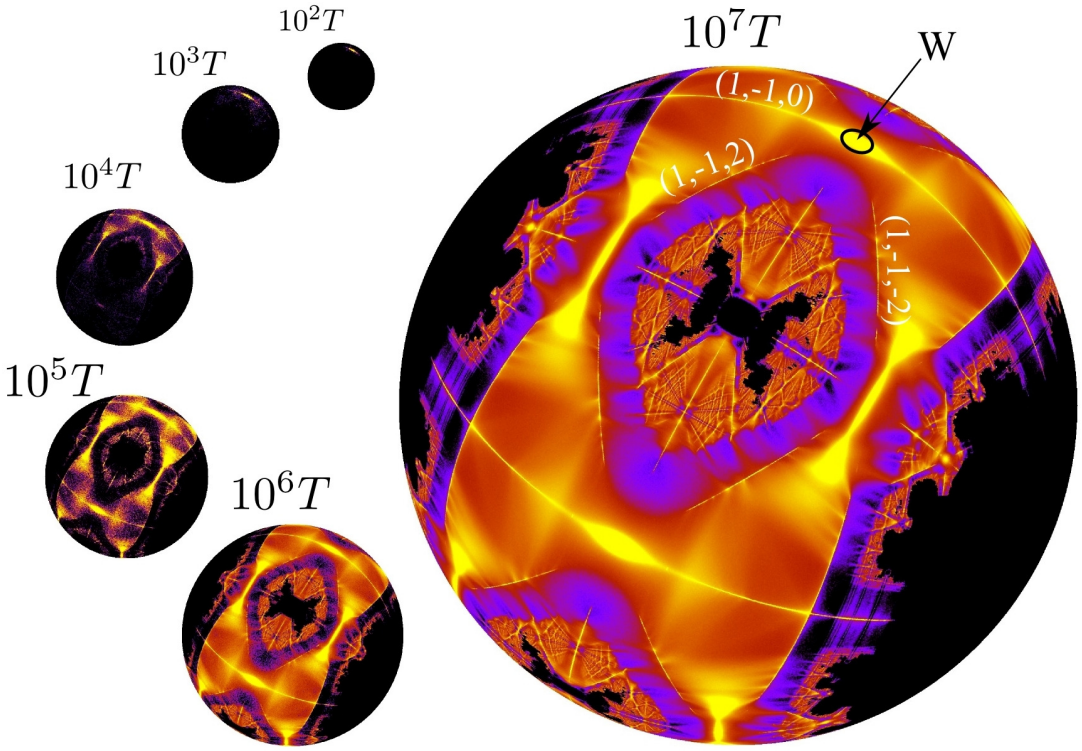


Figure 3.7.: Arnold diffusion of 32000 particles on the energy shell $E = 15$ of system 3.16. The effective perturbation strength is $\varepsilon_{\text{eff}} = 0.13$. The vectors $\mathbf{k}$ in white fulfill $\mathbf{k} \cdot \mathbf{p} = 0$ for all momenta on the resonance line.

e.g. $\mathbf{P}_{1,0,0} = (\sqrt{2E}, 0, 0)$ are not reached. The "forest" of KAM tori around them is too dense, especially for the z direction, where the winding number generated by the momentum is twice that of the other directions due to the potential. The calculation of this very simple system already takes a huge time.

If our aim is just to see the phase space structure, we can distribute the initial conditions over the whole energy shell to see the AW. This is done in Fig. 3.8. For resolving the fine structure close to the stable resonance $\mathbf{P}_{1,0,0}$ the averaging interval Δt is increased. Around the double resonance $\mathbf{P}_{0,0,1} = (0, 0, \sqrt{2E})$ which is more stable the distribution is almost as uniform as its initial values although it is intersected by some single resonance lines. This is the case

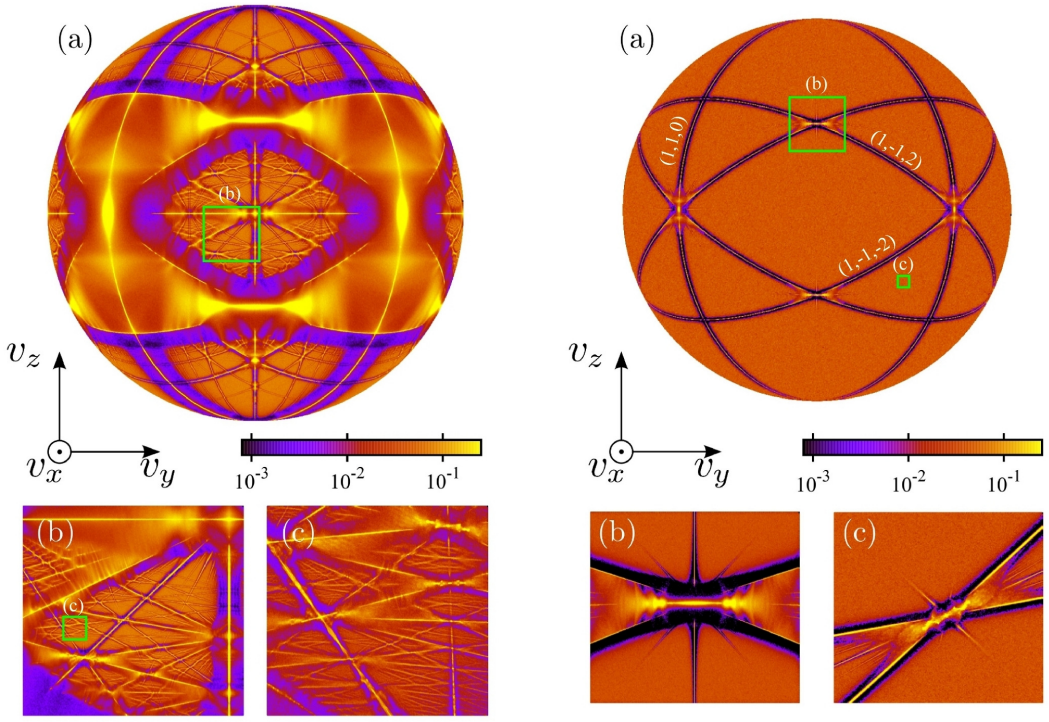


Figure 3.8.: Left: Probability density of the average speed of a microcanonical ensemble on the energy shell $E = 15 \Leftrightarrow \varepsilon_{\text{eff}} = 0.13$. For the two orders of magnitude zoom (c) an averaging time of $\Delta t = 200T$ was used to improve resolution. Right: Energy shell $E = 400 \Leftrightarrow \varepsilon_{\text{eff}} = 0.005$. Due to the smallness of the perturbation the distribution is almost homogeneous but a magnification to resonant points (c) reveals smaller resonances.

almost everywhere, if we move to a higher energy shell. Nevertheless, with enough magnification and integration time the resonance web can also be found everywhere, as shown in Fig. 3.8. The resolution of the AW via an initial momentum distribution over the whole energy shell takes only a few minutes on the GPU, while the simulation of the diffusion process from one point in momentum space is not finished after one day in the sense, that not the whole sphere is covered. So the question remains whether this system is ergodic.

3.4. The ratchet effect

This chapter analyzes, how AD can be utilized for transport in the system. It is easy to generate a directed motion from a nonvanishing average force. If one only has a force which is zero on average but is still fluctuating (randomly or deterministically) one has to employ a symmetry breaking machinery which profits more from the positive parts of the force than it suffers from the negative ones. This machine is called a ratchet (see Ref. [HM09] for a recent review).

Being so simple the ratchet is almost ubiquitous in nature. Actually it is mainly studied in economics, where it appears e.g. in an employee's increasing spendings if his income is consequently raised and lowered, and it is believed to be the mechanism for inflation, cf. [Gol77]. In physics the symmetries can be stated as follows: If the differential equation a systems obeys is invariant under one of the symmetry transformations [FYZ00]

$$S_1 : (\mathbf{r}, t) \to (-\mathbf{r} + \mathbf{r}', t + t) \tag{3.18}$$

$$S_2 : (\mathbf{r}, t) \to (\mathbf{r} + \mathbf{r}', -t + t), \tag{3.19}$$

which means basically that it is symmetric to some point in space or time, then one finds a second trajectory from a given one by applying this transformation to the trajectory. Since the velocity changes sign under each of the two transformations

$$S_i(\mathbf{v}) = -\mathbf{v}, \;\; i = 1, 2, \tag{3.20}$$

the speed of this symmetric trajectory will cancel the speed of the initial one and the average speed of all trajectories will be zero. So far mainly one dimensional ratchets have been investigated, but also some work has been done on two dimensions [SKH$^+$09, SSPRG03, DZFY08].

We have seen that in the chaotic regime with 2 DOF full control of the system is only possible in the regular part of the phase space. The only thing we can rely on for a chaotic trajectory is ergodicity. The system will cover the chaotic layer between the bounding KAM tori uniformly in time. Though in practice this averaging time can be quite long due to stickiness to some

manifolds which leads to Levy flights. So the area of the chaotic layer like that in Fig. 3.1 determines the long time average velocity of every trajectory in it via [SDK05]

$$v_{\mathrm{ch}} = \frac{\langle T\rangle_{\mathrm{u}} - \langle T\rangle_{\mathrm{l}} - \sum_i A_i w_i}{A_{\mathrm{layer}} - \sum_i A_i} \tag{3.21}$$

where $\langle T\rangle$ is the kinetic energy (of the upper and lower KAM torus surrounding the layer), w_i are the winding numbers of regular islands submerged in the chaotic layer and A_i their area. A_{layer} is the phase space area of the whole chaotic layer (including stable islands).

It would be nice to have such a simple rule for higher dimensional systems. But as we have pointed out, AD leads to penetration of any border, there is no asymptotic regime and ergodicity is actually a very tough question. But for a first numerical study there are two standard types of ratchets. The flashing ratchet

$$V_{\mathrm{flash}}(\mathbf{r}, t) \propto A(t)W(\mathbf{r}) \tag{3.22}$$

and the rocking ratchet

$$V_{\mathrm{rock}}(\mathbf{r}, t) \propto W(\mathbf{r} - \mathbf{A}(t)) \tag{3.23}$$

where $A(t)$ is a periodic function of time and $W(\mathbf{r})$ is periodic in space. With both flashing or rocking symmetries can be broken and a ratchet current can be produced. In Fig. 3.9 we report the speed distribution for a rocked system in the potential

$$V(x, y, t) = \cos(x)[1 + \cos(2y)] + 3x\cos(2t) + 3.5y\sin(t), \tag{3.24}$$

in which the symmetry breaking is done via a biharmonic driving.

The distribution is asymmetric and a ratchet transport occurs. The resonance peaks have not changed their position, just the 'flesh' around them has moved. Nevertheless the quality of the transport is not good since the width of the distribution grows much faster than its mean. The symmetry analysis only

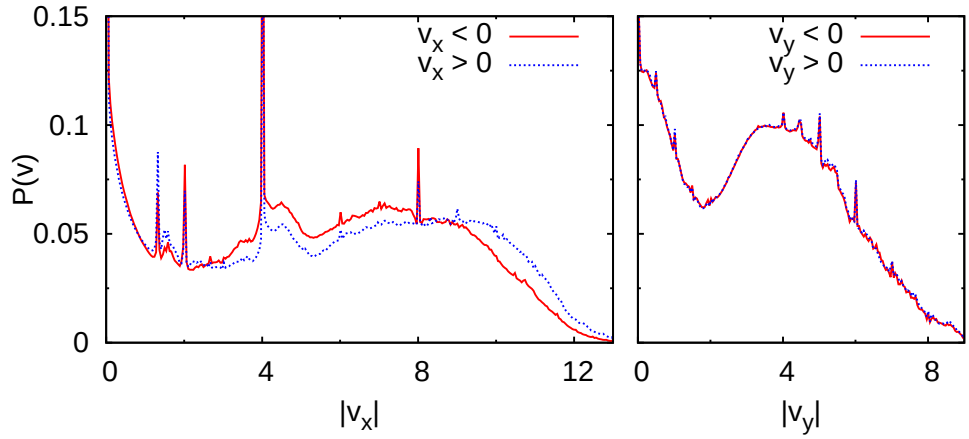


Figure 3.9.: Marginal distributions of average speed at time $t = 10^6$ (negative velocities in red) show perfect symmetry in y direction because the system is invariant under $y \to -y \wedge t \to t$ but for the x coordinate no symmetry holds.

predicts whether there is a current, but tells nothing about its strength. All efforts to improve this current by adjusting the parameters were not successful. The current was always small in comparison to the diffusion. One reason for this from the view of the AD is the small number of resonances to couple and a sharp slow down of AD. Therefore we will switch to a more promising ratchet in the next section.

3.5. The gating ratchet

The gating ratchet combines flushing and rocking [BM05, SMHN04]. Its potential is

$$V_{\text{gating}}(x,t) = A[1-\cos(\omega t)]\cos[x - B\sin(\omega t)]. \tag{3.25}$$

From an engineering point of view a gating ratchet is a pump or conveyor belt. Thus it is intuitive that it produces a directed current. The effect of the two parameters A and B on the Poincare section of the 1d ratchet is depicted in Fig. 3.10. The amplitude A controls the width of the chaotic layer. The rocking parameter B shifts the center of the layer and also increases its width.

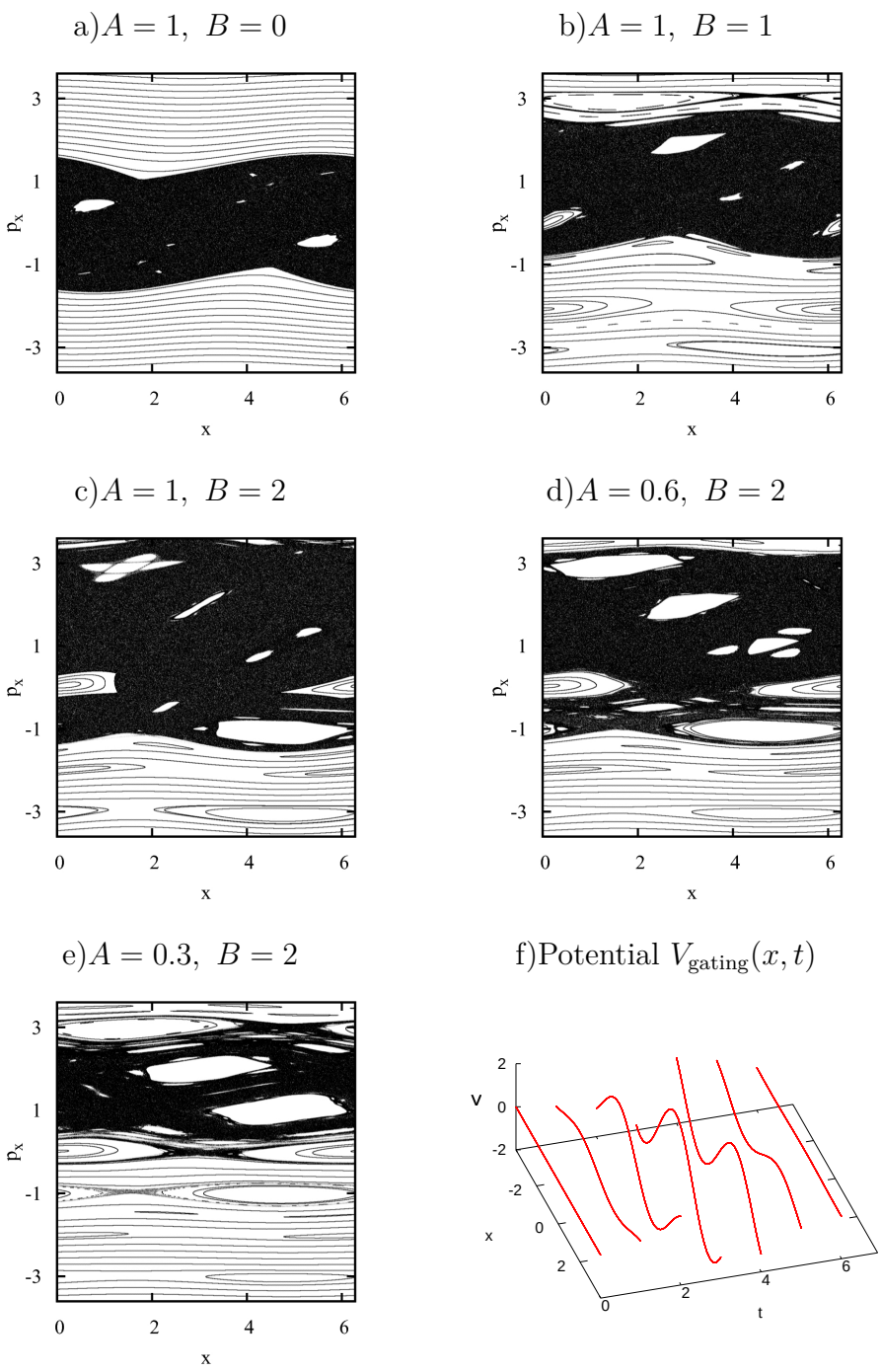


Figure 3.10.: Poincare sections of the 1D gating ratchet system $H = \frac{p^2}{2} + V_{\text{gating}}(x,t)$ for increasing rocking a-c) and decreasing amplitude c-e). f) The gating potential $V_{\text{gating}}(x,t)$.

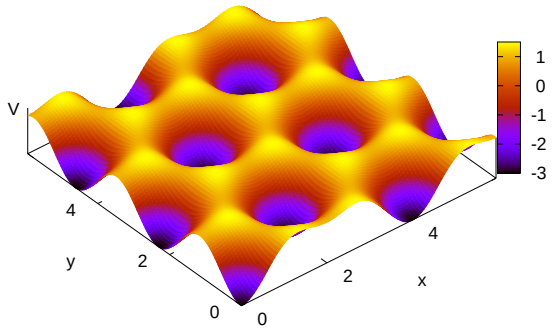


Figure 3.11.: The honeycomb potential $V_{\text{honey}}(x, y)$ 3.26

By controlling both parameters the chaotic layer can be brought to any position maintaining its width.

In this one dimensional system a directed asymptotic current can be achieved. But the initial conditions have to be placed inside the chaotic island. For initial conditions with vanishing initial momentum, the current will also be almost zero for all time.

With help of the AD this can be overcome. Noticeable in the Poincare sections Fig. 3.10 is the much richer structure of the shifted layer which is due to the combination of many harmonics in the driving. This also promises a faster AD for the 2D system. By adding one DOF the web opens a possibility for particles to diffuse from the origin to this chaotic sea and a current emerges from an ensemble initially at rest. For this purpose we will employ a honeycomb potential

$$V_{\text{honey}}(x, y) = -\left[\cos(\frac{\pi}{2}x + \frac{\pi\sqrt{3}}{2}y) + \cos(\frac{\pi}{2}x - \frac{\pi\sqrt{3}}{2}y) + \cos(\pi x)\right]. \quad (3.26)$$

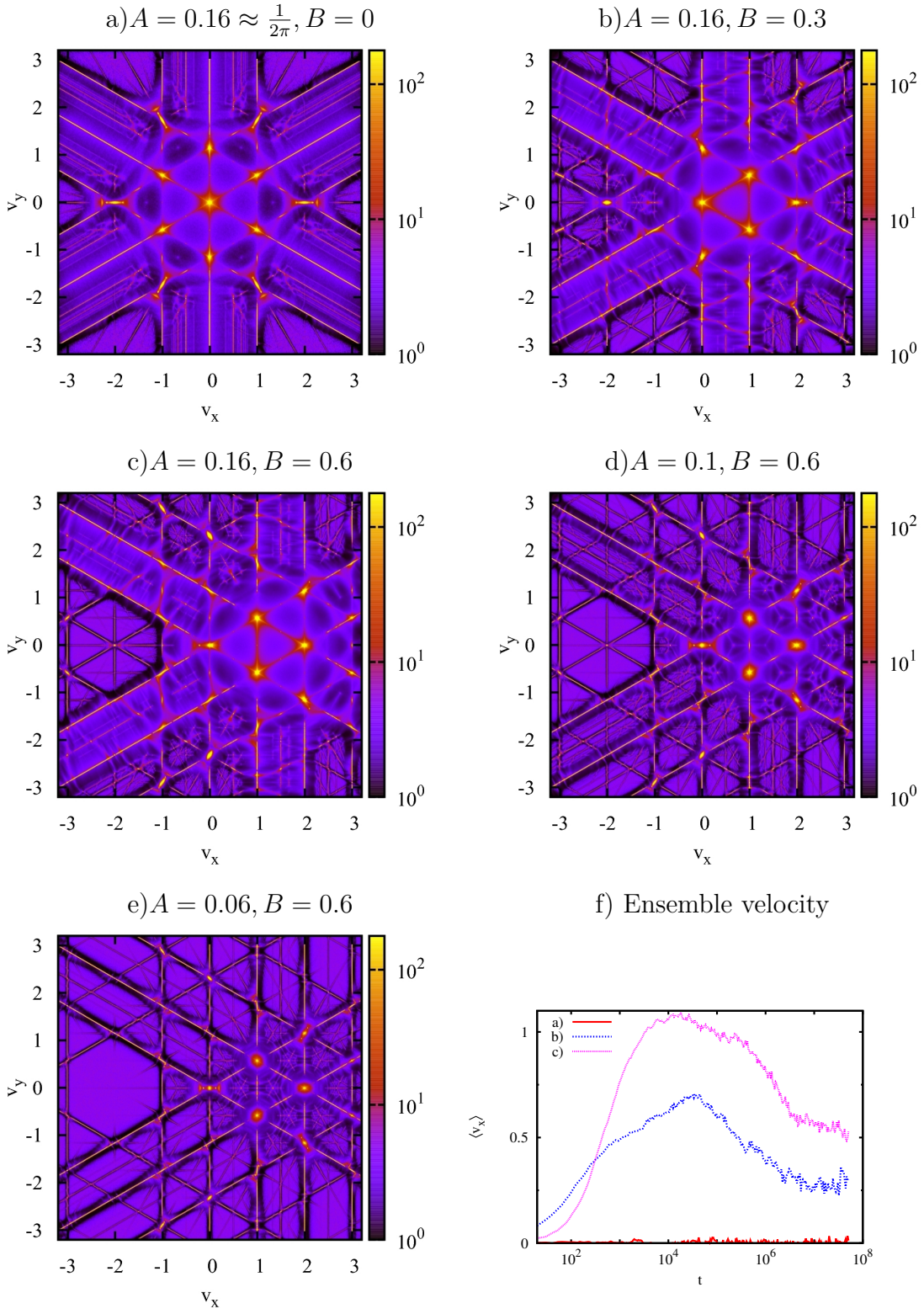


Figure 3.12.: Adjusting the Arnold web of system Eq. 3.27 to a desired center a,b,c) and width c,d,e). f) Ensemble velocity $\langle v_x \rangle$ for the first three sets of parameters (stronger rocking) for an ensemble initially at rest $\mathbf{p}(t = 0) \approx 0$.

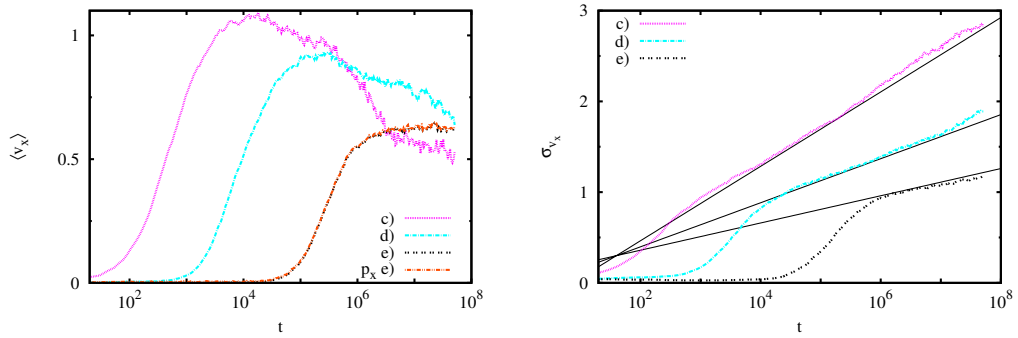


Figure 3.13.: Mean velocity $\langle v_x \rangle$ and standard deviation σ_{v_x} of an ensemble initially distributed close to $\mathbf{p}(t=0)=0$ for the last three sets of parameters c,d,e) in Fig. 3.12 (constant rocking and decreasing amplitude. For e) also the instantaneous momentum $\langle p_x \rangle$ is shown. Once the ensemble has entered the central part a logarithmic scaling $\sigma_{v_x}(t) \propto \ln t$ shows up.

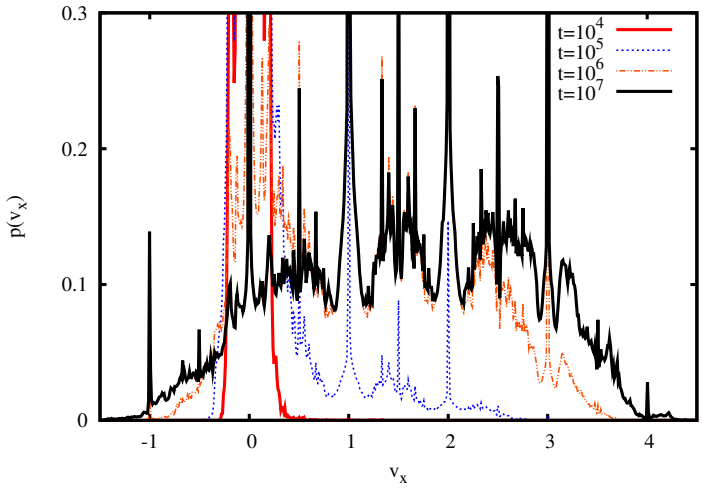


Figure 3.14.: Time evolution of the marginal distribution $W(v_x,t)$ for $A = 0.06, B = 0.6$ (the case case e) on Fig. 3.12). For $t = 10^4 T$ 66% of the ensemble is at the center resonance $v_x = 0$. This drops to 62%, 50%, 46% for $t = 10^5, 10^6, 10^7$. The rest of the ensemble gathers with 6% exactly at speed $v_x = 1$ and another 5% at $v_x = 2$.

3. Arnold diffusion

This is the simplest potential with 2 coupled DOF from a numerical and from the point of view of an application. Just three sin functions must be evaluated to calculate the force. Likewise, only three laser beams are needed to produce it, which makes it easy to control. Moreover, it shapes the most beautiful figures. We apply a gating procedure and end up with the potential

$$V_{2D,\text{gating}}(x, y, t) = A[1 - \cos(\pi t)]V_{honey}(x - B\sin(\pi t), y). \tag{3.27}$$

The average speed distributions for parameters similar to the 1d case are shown in Fig. 3.12. Mind the rescaling of time and space by a factor π. This keeps the momentum invariant. Also the amplitude of the honeycomb potential is approximately twice the amplitude of the 1d ratchet.

The method to steer a cloud at a given velocity is as follows. First one takes a flashing strength$A$ for which AD is present, i.e. the DOF are not too close and not too far from resonance as in the first web of Fig. 3.12. Then the AW must be shifted via the rocking parameter B, so that its center is at the desired speed. Consequently one adjusts the potential strength A so that the origin lies at the rim of the web. This makes it very likely for particles at rest to diffuse into the center of the AW. Fig. 3.12 shows that we have hereby produced a current which is constant over a long time span (99% of the time, mind the logarithmic timescale). This happened without any attractive solution as in the sense of a dissipative system. And the procedure can still be improved.

This is remarkable considering the chaotic nature of the system. The time evolution of the marginal speed distribution $W(v_x, t)$ is depicted in Fig. 3.14. The distribution is quite ordered. 50% of the ensemble are always at $v_x = 0$. The particles don't only stay there, they also come back once they have left the resonance. Regarding the remaining distribution approximately 6% went to the exact speed of $v_x = 1$ and another 5% went to $v_x = 2$. The next question

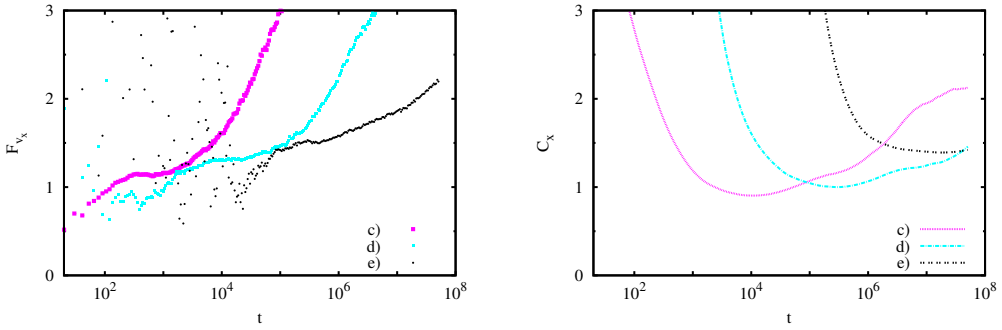


Figure 3.15.: Transport quality given by the Fano factor F_{v_x} of the velocity distribution and the coefficient of variation C_x of the distribution in space.

is about the quality of the transport. An appropriate quantity in this respect is the Fano factor [Fan47] or index of dispersion

$$F = \frac{\sigma^2}{\mu} \tag{3.28}$$

where σ^2 is the variance of a distribution and μ is its mean. The smaller the Fano factor, the better is our transport quality. Another common measure is the dimensionless coefficient of variation

$$C = \frac{\sigma}{\mu}. \tag{3.29}$$

We consider C_x for the distributions of the x coordinate of the particles and F_{v_x} for their average speed in Fig. 3.15. The coefficient C_x for the distribution of the x coordinate is already low. If one neglects the δ peak at the central resonance, the standard deviation is not changed but the mean doubles and C drops by a factor of 2. For a normal distribution a value of $C = 0.6$ means that only 5% of the area is on the negative part of the x axis. Or in other words 95% of our particles have taken the right direction. Certainly this process can be optimized by choosing other parameters A and B or by adding more laser beams and therefore adding higher harmonics that are centered at a more distant point in momentum space. Also the resonant velocities can be

extracted. The main obstacle is that the center resonance keeps most of the particles.

3.6. Nekhoroshev's estimates: The speed of Arnold diffusion

We placed the initial conditions close to the border of the AW. But indeed such a border is not to be defined exactly. The AW is rather ubiquitous. Only our observations suggests some confinement due to the timescale. On the other hand we saw a logarithmic growth of the standard deviation $\sigma_{p_x} \propto \ln t$ and a dependence of the transition time to enter the AW on the potential strength. It is a hard task to make quantitative statements about chaos in several dimensions which is the aim of this chapter. To be more precise, we are looking for a measure of the diffusion speed and the time to escape a full resonance. The only analytical thing to be said about AD are the Nekhoroshev estimates [Nek77, PÖ93] for the weak perturbation limit

$$H = H_0(\mathbf{p}) + \varepsilon H_1(\mathbf{p}, \mathbf{q}). \tag{3.30}$$

As in the case of AD, H_0 is integrable and H_1 is a nonlinear perturbation with ϵ measuring its strength. There is a threshold ϵ_0 below which we enter the Nekhoroshev regime. In this regime there are exponents a, b which determine an upper bound for momentum change in the following way

$$\exists \varepsilon_0, a, b : \forall \varepsilon < \varepsilon_0 \wedge \forall t < T_0 \exp(\varepsilon^{-a}) : |\mathbf{p}(t) - \mathbf{p}(0)| < P_0 \varepsilon^b. \tag{3.31}$$

The constants a and b depend on the number of DOF. The bigger they are, the stronger is the statement. An universal value for both is not obtained yet. [PÖ93] gives the simple global value $a = \frac{1}{2n}$ and even $a = \frac{1}{2}$ in the vicinity of a resonance. And $b = a$ and even $b = \frac{1}{2}$ not only close to a resonance but also in a larger domain. So the number of DOF gets measurable by the AD. In a driven system time counts as a half DOF.

So on the other hand the time T_* for the fastest trajectory to reach a certain

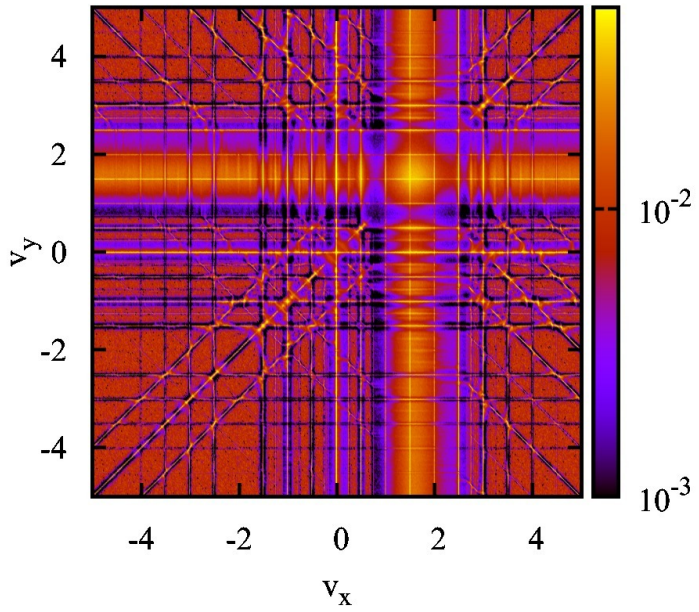


Figure 3.16.: Arnold web for the parameters $A = 0.25$, $\mathbf{B} = (0.6, 0.6)$, $\epsilon = 0.1$. The coupling ϵ is reduced later on to investigate the diffusion time from the origin. This will not alter the position of center of the web which is at $p = (1.5, 1.5)$ in this a priori chaotic system.

distance in phase space $max|\mathbf{p}(t) - \mathbf{p}(0)| =: \Delta p_{max}$ scales exponentially with the perturbation strength

$$T_* = T_0 \varepsilon^{-1} \exp\left(\varepsilon^{-a}\right). \tag{3.32}$$

The Nekhoroshev theorem also holds for a priori chaotic systems, i.e. $H_0(\mathbf{p}, \mathbf{q})$ is already chaotic but the DOFs are not coupled. We apply this to a potential

$$\begin{aligned} U(x, y, t) &= A(t)[\cos^2(x + \varphi_x(t)) + \cos^2(y + \varphi_y(t)) \\ &+ \epsilon \cos(x + \varphi_x(t)) \cos(y + \varphi_y(t))] \end{aligned} \tag{3.33}$$

which is constructed by 4 laser beams, two of which form a standing wave in either x or y direction. Their polarization determines the coupling ϵ. If both beams are polarized in the same direction, the interaction is maximal with $\epsilon = 2$. If they are perpendicular, the interaction vanishes $\epsilon = 0$ [GR01]. This system is a priori chaotic without any interaction. So resonance channels

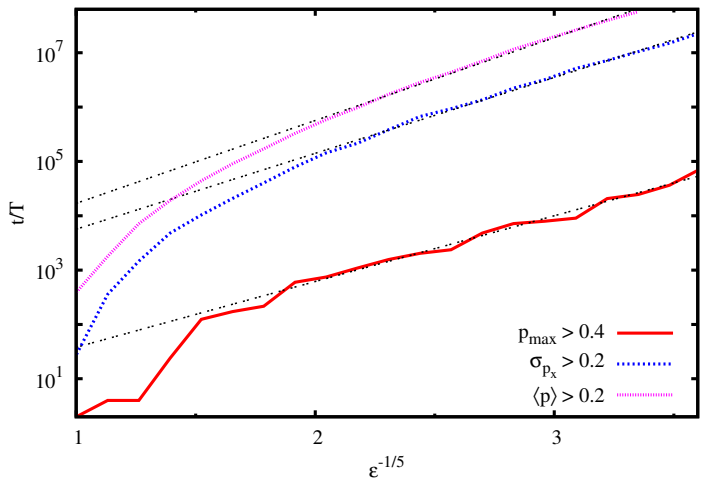


Figure 3.17.: Acceleration time t for the highest velocity p_{max}, the standard deviation σ_{p_x} and the average momentum $\langle p \rangle$ of the ensemble to reach a certain threshold for several values of ϵ and one parameter fit $t = a \exp(b\epsilon^{-1/5})$. All of them are determined by Nekhoroshev's scaling. Gating parameters are $A = 0.25$, $B = (0.6, 0.6)$ as in Fig. 3.16.

are already there. There is just no AW to connect them. Upon increasing ϵ, the DOF couple the chaotic regions are connected and global diffusion in momentum space can occur.

The driving obeys a gating protocol with $A(t) = A(1+\cos(\pi t))$ and the phase shift $\varphi_i(t) = B_i \sin(\pi t)$. Thus, the rocking parameter $\mathbf{B} = (B_x, B_y)$ determines the direction of the transport or rather the center of the stochasticity in the p_x-p_y-plane. In Fig. 3.16 we choose the parameters A and $\mathbf{B}$ so that this center is well distant from the origin at $\mathbf{v} = (1.5, 1.5)$. So an ensemble at rest should be accelerated towards this point. In Fig. 3.17 the time for the fastest particle, the standard deviation and the average velocity to reach a certain value is depicted as a function of the coupling ϵ.

The Nekhoroshev estimate is an upper bound for any trajectory. But as the figure shows, it determines the time for a maximal momentum $T^*_{p_{max}}(\epsilon)$ quite exactly. It is remarkable that this formula also determines the behavior of the standard deviation of an ensemble as we observed in Fig. 3.12. The time $T^*_{\sigma_p}(\epsilon)$ to reach a certain RMS in momentum space is also $\propto \exp\left[\left(\frac{\epsilon_0}{\epsilon}\right)^{0.2}\right]$. Also the

scaling for the time T_* for the fastest particle to reach a certain distance in action space is the same as for the ensemble average to reach a certain speed.

The fits suggest the scaling

$$T^*_{p_{max}}(\epsilon) \propto T^*_{\sigma_p}(\epsilon) \propto T^*_{\bar{p}}(\epsilon) \propto \exp\left[\left(\frac{\epsilon_0(p)}{\epsilon}\right)^{0.2}\right] \tag{3.34}$$

where p stands for the three values p_{max}, σ_p, $\bar{p}$. So far Nekhoroshev's theorem was mainly studied with symplectic maps. These maps correspond to a delta kicked Hamiltonian which does not fulfill the smoothness condition and a diffusion could occur without an Arnold mechanism. Nevertheless as the numerical errors in the appendix A show, the number of time steps can be drastically decreased in a simulation, recovering a map from an originally smooth system.

3.7. Is Arnold diffusion a diffusion?

In spite of its title, the Arnold diffusion is rarely investigated as a diffusion. On the one hand, the author of [Loc99] states that "the word 'diffusion' is in fact misleading, and perhaps not without a tint of wishful thinking". This is too strict, since diffusion simply means spreading, which certainly occurs. On the other hand, according to [EH13], the AD is everywhere locally normal diffusive. But locally is hard to define in this fractal landscape, which the author also points out. So the features of diffusion are reviewed here.

Diffusion refers to a process of particles spreading in space. The most prominent example is the Brownian motion [Bro28], which results in the normal diffusion, i.e. the mean square displacement (MSD) grows linear in time

$$\langle \sigma_x^2(t) \rangle = 2Dt \tag{3.35}$$

with some diffusion constant D. Diffusion processes are usually modeled with stochastic forces. If there are long time correlations in the system, the time evolution can follow a different power law

$$\langle \sigma_x^2(t) \rangle = 2Dt^\alpha \tag{3.36}$$

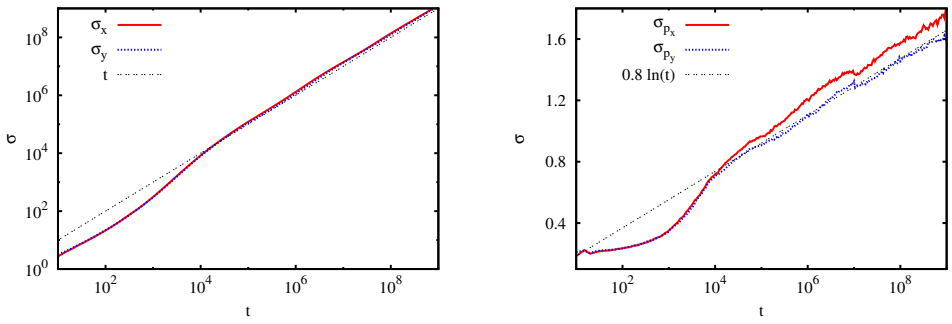


Figure 3.18.: Standard deviation of space coordinates (left) and momenta (right) for particles in the potential Eq. 3.37. Diffusion in real space is slightly superballistic $\sigma_x/t = \mathcal{O}(\ln t)$. Due to the coupling of many harmonics in this system diffusion in momentum space does not only happen on a logarithmic timescale but even fits $\sigma_p = \alpha \ln t$ quite exactly. Generally only $\sigma_p = \mathcal{O}(\ln t)$ holds.

where the regime of $\alpha < 1$ is called subdiffusive and $\alpha > 1$ is superdiffusive. Other reasons for anomalous diffusion can be that the system is not in equilibrium, or that the space accessible for the particles has an anomalous structure, like a fractal or a space with curvature. A good quantity that describes diffusion processes is the so called propagator $W(r,t)$ which is the probability density to find a particle at point r after some time t. One could also calculate the probability for the velocity $W(v,t)$.

AD involves the whole phase space and is therefore a rather mathematical diffusion. As it could be shown in this chapter, it is something in between real space and velocity space. The AD occurs in deterministic Hamiltonian systems, where due to nonlinearity chaotic forces replace the stochastic ones. To compare it with the diffusion of pollen in water, the action variables take the role of the position of the pollen, and the weak but fast switching nonlinear forces, generated by the angle variables, compare to the random forces of the surrounding water molecules. For the AD a separation of timescales shows up and a distribution of average velocity $W_\tau(\bar{v},t)$ where the average is taken over some smaller time $\tau \ll t$ bears more information about what happens than a distribution of instantaneous coordinates. For the AD to occur the system needs more than two DOF and must be nonlinear but close to integrable. As

it appeared in this chapter, this is almost always the case in the long run. Almost no realistic system is linear and has one or two DOFs. Those are just the systems that we can observe, since they are the most stable ones and (maybe for the same reason) that we can compute easily. Most systems consist of many particles, are coupled to some environment and therefore have a lot of DOF. On the other hand most systems of our interest don't exhibit a strong chaoticity. A reason for this can also be taken from the calculations in this chapter. If the phase space consists of strongly chaotic and almost integrable regions, we saw that per definition strong chaos repells the particles and they stick close to integrable regions.

Also the AD slows down exponentially in the vicinity of KAM tori and is numerically an exponential problem. So it can be concluded that AD happens at the border of what is computable in Hamiltonian dynamics. To answer the question in the title, AD in the physical systems investigated here is subdiffusive ($\alpha < 1$) in the velocity space and superdiffusive ($\alpha > 1$) in real space. A driven system has even a diffusion exponent $\alpha > 2$ since there is no dissipation but continuous energy pumping. This is termed hyperdiffusive or superballistic. But certainly some other cases can be constructed on intermediate timescales. The overall average speed and the instantaneous momentum assume almost the same profile which is only possible since the time a particle spends close to its new speed after an acceleration is exponentially larger than the time before. In Fig. 3.18 the MSD in coordinate and momentum space for the system

$$V(x, y, t) = A[1 - \cos(\pi t)]W_{honey}(x - B\cos(\pi t), y). \tag{3.37}$$

is shown as example.

3.8. Computation of diffusion coefficients

Although we find that the diffusion in momentum space is not at all locally normal diffusive, we naively define a diffusion constant as if it were so via the MSD of particles with initial momentum $\mathbf{p}$ after a time Δt.

$$D^{\Delta t}(\mathbf{p}) = \frac{\sigma_p^2(\Delta t)}{\Delta t} \tag{3.38}$$

So this diffusion coefficient depends on momentum which is clear. But it will also depend on the time Δt. Since the time seems to obey the Nekhoroshev law $T^*_{\sigma_p}(\epsilon) \propto \exp \epsilon^{-a}$, it is expected that for a fixed standard deviation, i.e. a fixed definition of locality, the diffusion coefficient scales as

$$D = \frac{\sigma_p^2}{T^*_{\sigma_p}(\epsilon)} \propto \exp\left(\epsilon^{-a}\right). \tag{3.39}$$

This is the diffusion coefficient observed in [GL13, FGL05, LGF03].

We can also assume that the RMS σ_p of the distribution follows the Nekhoroshev law $\sigma_p = P_0\epsilon^b$. Inserting the effective perturbation strength $\epsilon_{eff} = \frac{\epsilon}{p^2}$ which is momentum dependent, we end up with a diffusion coefficient

$$D(p,\epsilon) = D_0 \left(\frac{\epsilon}{p^2}\right)^{2b} \exp\left[-\left(C\frac{p^2}{\epsilon}\right)^a\right]. \tag{3.40}$$

For a global diffusion the time exponent $a = \frac{1}{2n}$ is the decisive one whereas the momentum exponent can be assumed to be $b = \frac{1}{2}$. Although $b = a$ in some small domain, this would lead to a faster diffusion in those domains, repelling the trajectories towards the slower domains with $b = \frac{1}{2}$. Thus only 2 unknowns remain, which are the scaling parameter C and the diffusion strength D_0.

For big values of ϵ ($\epsilon = 0.1$ like in Fig. 3.12 is a big value for the Arnold diffusion) we almost don't feel the exponential. So we are left with a diffusion coefficient

$$D(p,\epsilon) = D_0\frac{\epsilon}{p^2}. \tag{3.41}$$

This diffusion can be approximated by considering that a discretization of the normal diffusion equation with constant D in both variables with $\frac{\Delta p^2}{2\Delta t} = D$ leads to a random walk. Thus our p dependent diffusion coefficient leads to a hopping on a nonequidistant lattice in which the step length Δp scales as

$$\Delta p(p) = \sqrt{2D(p,\epsilon)\Delta t} = \frac{\sqrt{2D_0\epsilon\Delta t}}{p}. \tag{3.42}$$

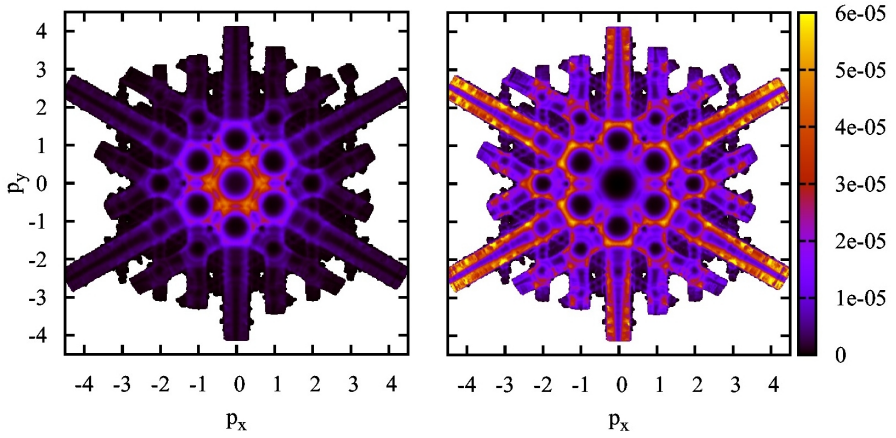


Figure 3.19.: Left: Diffusion coefficients $D_{1000T}(p)$ calculated via an ensemble initially at $p = (0, 0)$, i.e. inside the Arnold web. Right: The same data multiplied with p^2. The potential is given in Eq. 3.37.

So the distance of the nth point (to the right) is

$$p(n) = \sum_{k=1}^{n} \frac{p_0}{k} \approx p_0 \ln n \tag{3.43}$$

where $p_0 = \sqrt{2D_0\epsilon\Delta t}$. Since we are not concerned on the origin which should be smooth anyway, we prefer the symmetric function

$$p(n) = p_0 \sinh^{-1}(n) \tag{3.44}$$

Any invariant distribution occurring on the n lattice

$$W_n(n,t) = \frac{1}{t^\alpha} f\left(\frac{n}{t^\alpha}\right) \tag{3.45}$$

would lead to a MSD for long times $t \gg 1$

$$\sigma_p^2(t) \;=\; \int_{-\infty}^{\infty} \left[p_0 \sinh^{-1}(n)\right]^2 W_n(n,t)dn \approx (p_0\alpha \ln t)^2$$

which is the observation of Figs. 3.13 and 3.18.

4. Summary and outlook

This thesis illustrates the efficiency of computation on a GPU with the example of the Arnold diffusion, which has been explored in unprecedented detail. The Arnold diffusion appears in near integrable Hamiltonian systems and demands the evaluation of huge statistical data sets. The obstacle is the occurrence of rare events like the entering of particles into narrow resonance channels. This determines the long time behavior and the global diffusion which 50 years after its finding [Arn64] is still lacking a quantitative results. The corresponding systems show a behavior seemingly uncorrelated on short and long timescales. With raw computational power it is possible to track both timescales without any different treatment. Although a universal diffusion model is still missing, in a subcase of a homogeneous resonance web a simple relation showed up. So the attempt to build up a diffusion model was made. This finding, which was preceded by scanning a big set of parameters and models, was only possible through the speedup factor of 100 through the parallelization with CUDA. The noisy phase dynamics in Josephson junctions and the Kuramoto model have been studied via GPU numerics [JK10], as well as inertial Brownian motors driven by colored noise [KLH09] and the evolution of nonlinear lattices [ZDH11]. But the breakthrough of GPU computing at least in nonlinear dynamics is only about to come, maybe even in quantum mechanics.

The present approach to the visualization of the Arnold web bears advantages with respect to theory and experiments in comparison to the methods in [FGL00, MPV86, TK07, DL93], which demand pre- or after-processing of the data. The average speed distribution needs only straightforward integration of the equations of motion. Thus, it can directly be extended to the quantum limit while it is not at all clear how the concept of finite-time Lyapunov exponents [FGL00, MPV86, TK07] or the frequency analysis [DL93] should be

generalized to the Schrödinger equation.

Ultracold atoms offer the possibility for the experimental realization of our method. Today the creation of three-dimensional periodic optical potentials is an experimental routine in many labs [MO06]. Additionally, the driving and a confinement to two dimensions constitutes no obstacle. With the Bragg selection technique [SKH$^+$09, SIC99], a diluted Bose-Einstein-Condensate can be prepared in the necessary initial state. The characteristic decoherence time is much larger than the period of the oscillations at the bottom the potential well. This guarantees Hamiltonian evolution. The velocity of atoms can be tracked by time-of-flight measurements. Since quantum effects play an essential role in the system's evolution, the transition from the classical to the quantum regime can be examined.

A. General integrator for almost everything

This appendix aims at a method that should be known by every numerical physicist. Luckily it is already known to most of them but without awareness of its broad applicability because of the different names this method has. The physical or mathematical reason for its usage is varying, but the general idea and the numerical implementation is the same. In classical Hamiltonian systems it's called symplectic integrator [Yos90], in many body quantum physics it's called Suzuki [Suz76] or Trotter [Tro59] Formula. The term I want to use because of neutrality and because it catches the point is the split operator method [FMF76] used mainly in chemical physics to integrate the space representation of the Schrödinger equation [BCM06]. Using the notation of the Poisson bracket for classical mechanics

$$\{f, g\} := \sum_i \left(\frac{\partial f}{\partial q_i} \frac{\partial g}{\partial p_i} - \frac{\partial g}{\partial q_i} \frac{\partial f}{\partial p_i} \right) \tag{A.1}$$

the time derivative of every point $z = (q, p)$ in phase space reads

$$\dot{z} = \{z, H\} . \tag{A.2}$$

Defining this Poisson bracket with the Hamilton function as an operator

$$D_H := \{\, . \,, H\} \tag{A.3}$$

the classical time evolution can be written as

$$z(\tau) = e^{\tau D_H} z(0) \tag{A.4}$$

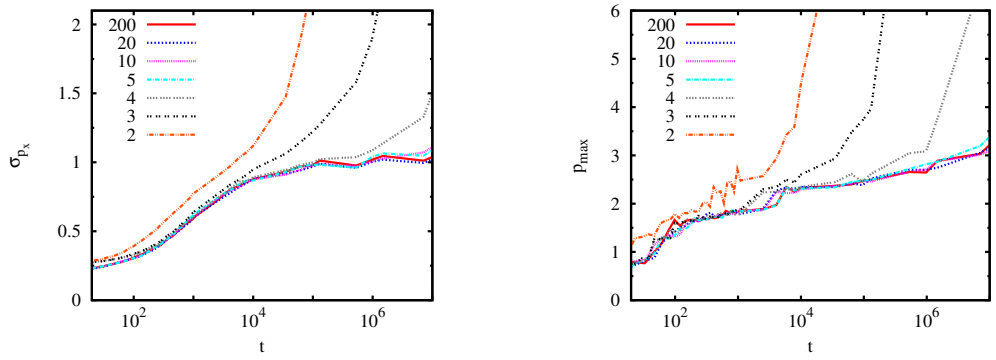


Figure A.1.: Standard deviation of momentum σ_{p_x} (left) and momentum p_{max} of the fastest particle for the Honeycomb system ($\epsilon = 0.05$) computed with the second order integrator $SABA_2$. The color indicates the number of steps. One can cleary see that in the considered time 5 steps are completely sufficient and deliver both values of the distribution correctly. The time for reliable calculations seems to grow exponentialy with the number of steps.

like the propagator of a Schrödinger equation with a Hamiltonian $\hat{H}$ where it would be

$$\psi(\tau) = e^{-i\tau\hat{H}/\hbar}\psi(0). \tag{A.5}$$

Using the Baker-Campbell-Hausdorff (BCH) formula for general linear operators A, B

$$e^A e^B = e^{A+B+\frac{1}{2}[A,B]+\frac{1}{12}([A,[A,B]]+[B,[B,A]])+\ldots} \tag{A.6}$$

several times for a Hamiltonian $\hat{H} = \hat{H}_A + \hat{H}_B$ a propagator for a small timestep τ can be approximated by a product

$$\exp\left(i\tau\hat{H}\right) = \prod_{j=1}^{k} \exp(ic_j\tau\hat{H}_A)\exp(id_j\tau\hat{H}_B) + \mathcal{O}(\tau^{1+n}) \tag{A.7}$$

for proper coefficients c_i, d_i which can be taken e.g. from [Yos90] or [FR90]. Since the Poisson bracket is linear in H, i.e.

$$D_{H_A+H_B} = D_{H_A} + D_{H_B} \tag{A.8}$$

the expression holds in the classical case

$$\exp\left(\tau D_H\right) = \prod_{j=1}^{k} \exp(c_j \tau D_{H_A}) \exp(d_j \tau D_{H_B}) + \mathcal{O}(\tau^{1+n}) \tag{A.9}$$

This can be numerically exploited in two ways. First one should decompose every Hamiltonian or Hamilton function into two terms which are easier to integrate. And second one can numerically keep the special structure of the differential equation which is the symplecticity in the classical case and the unitarity in the quantum regime which is not only more physical but also reduces the error. The propagator of the kinetic energy $H_A = T = \frac{p^2}{2m}$ is simply

$$\exp(\tau D_T) = \begin{pmatrix} \tau \frac{p}{m} & 0 \\ 0 & 0 \end{pmatrix} \tag{A.10}$$

and that of a potential only $H_B = V(x)$

$$\exp(\tau D_V) = \begin{pmatrix} 0 & 0 \\ 0 & -\tau V'(x) \end{pmatrix}. \tag{A.11}$$

In practice only fourth and six order order methods play a role if not even second order is sufficient. Higher orders don't seem necessary for time dependent problems since the order of the error for a set of coefficients c_i, d_i is only valid for constant Hamiltonians. Thus the error in the time dependence is always of order $\mathcal{O}(\tau)$. For most constant Hamiltonians the diagonalization should be preferred to a direct time integration.The split operator method outperforms the standard Runge-Kutta integrator for two reasons. First one gets a high order method by calculating very small integrators for subsystems. Second one can keep the symplectic or unitary structure of the ODE exactly (up to machine precision) since the smaller timesteps can easily be made symplectic or unitary.

For systems where one integrable part B is just a small perturbation of the integrable Hamiltonian A

$$H = A + \epsilon B \tag{A.12}$$

like it is the case for th Arnold diffusion a different type of integrator, which has not gained much attention, should be used which exploits the smallness of ϵB. The integrators can be found in [LR01]. The second order integrator $SABA_2$ leaves an error of order $\mathcal{O}(\tau^4\epsilon + \tau^2\epsilon^2)$ thus lowering the error of a standard symplectic integrator by a factor ϵ or τ^2.

B. Possible experimental setup - cold atoms in optical lattices

Ultracold atoms in optical lattices is the testbed for Hamiltonian systems [LR09]. With three or four laser beams all the potentials in this thesis can be generated including time dependence. This short introduction is based on the advisable report [GR01]. The electric field of a laser beam propagating with wave vector $\mathbf{k}$ in the xy-plane and polarized in z direction is the solution of the Maxwell equations

$$\mathbf{E}(\mathbf{r},t) = E_0 \mathbf{e}_z \Re\left\{\exp\left[-i(\omega t - \mathbf{k}\cdot\mathbf{r} - \phi)\right]\right\} \tag{B.1}$$

where ω is the laser frequency and ϕ is its phase. The electron (mass m_e) of an atom placed in this field feels the electric force $F = q_e\mathbf{E}(\mathbf{r},t)$. The motion of the whole atom with mass $m_a \approx 10^4 m_e$ happens on a much larger timescale. The potential it feels is that of a dipole $\mathbf{d} = \Re\left\{\varepsilon_0\alpha_0\mathbf{E}\right\}$ depending on the polarizability of the atom which is [TRG89]

$$\alpha_0 = -\frac{q_e^2}{2m_e\omega\varepsilon_0}\frac{\Delta - i\frac{1}{2}\Gamma}{\Delta^2 + \frac{1}{4}\Gamma^2} \tag{B.2}$$

with the detuning $\Delta = \omega - \omega_0$ from resonance frequency ω_0 of the atom and the classical radiative width Γ. The dipole force is $\mathbf{F} = -\mathbf{d}\cdot\nabla\mathbf{E}$. Its average over many oscillations of the laser field finally gives the potential

$$U(r) = -\frac{\varepsilon_0}{4}\alpha_0' E_0^2(r) \tag{B.3}$$

felt by the whole atom. With four laser beams one can produce the potential

$$\begin{aligned} U(x,y,t) &= A(t)[\cos^2(x+\varphi_x(t)) + \cos^2(y+\varphi_y(t)) \\ &+ \epsilon\cos(x+\varphi_x(t))\cos(y+\varphi_y(t))]. \end{aligned} \tag{B.4}$$

$A(t)$ is the amplitude of all 4 beams. $\varphi_x(t)$ is the phase modulation of the standing wave in x direction (same for y). And ϵ is determined through the angle between the polarizations of the standing wave in x and that in y direction. This ϵ is the perturbation parameter occuring in Nekhoroshev's formula.

C. Normal diffusion in a curved space

A normally diffusive process spreads like

$$\langle \Delta x^2 \rangle \propto t \tag{C.1}$$

The question to be answered here is, how do we have to curve the space to arrive at the diffusion law of Arnold diffusion

$$\langle \Delta y^2 \rangle \propto \ln^2 t \tag{C.2}$$

So let's keep everything one dimensional. The problem is to identify points in x-space with points in y space via a function $y = f(x)$ and we want a process that yields a Gaussian spreading in the flat x-space

$$p_x(x,t) = \frac{1}{\sqrt{2\pi t}} \exp \frac{-x^2}{2t} \tag{C.3}$$

to have a logarithmic spreading in y space

$$\begin{aligned} \langle \Delta y^2 \rangle &:= \int_{-\infty}^{\infty} y^2 p_y(y,t) dy \\ &= 2 \int_0^{\infty} f^2(x) p_x(x,t) dx \stackrel{!}{=} \ln^2 t. \end{aligned} \tag{C.4}$$

Both distributions $p_x(x,t)$ and $p_y(y,t)$ should be symmetric and initially concentrated at $x = 0$ and $y = 0$. So we end up with a Fredholm integral equation of the first type C.4 which is generally difficult to solve. But for our case the Gaussian kernel can be transformed into a Laplace transform via $z := x^2/2$

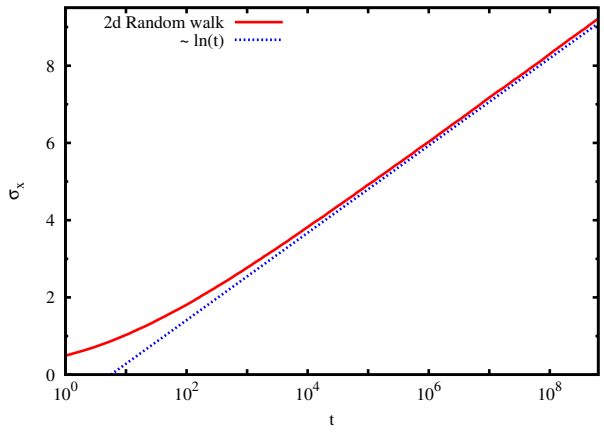


Figure C.1.: Random walk on a 2d lattice in which the distance of the nth bond from the origin is $x = \sinh^{-1}(n_x)$ thus the bondlength is $\Delta x(n_x) = \partial_{n_x} x \approx 1/n_x$ (the same applies for the y coordinate).

and $s = 1/t$

$$\int_0^\infty \underbrace{\frac{f^2(\sqrt{2z})}{\sqrt{2z}}}_{=:h(z)} e^{-sz} dz \stackrel{!}{=} -\sqrt{\frac{\pi}{2s}} \ln^2 s \tag{C.5}$$

So $h(z)$ is the inverse Laplace transform of the r.h.s. which is

$$h(z) = \frac{[\lambda - \ln(z)]^2 - \frac{\pi^2}{2}}{\sqrt{\pi z}} \tag{C.6}$$

where λ is some constant. On long distances we get the simple dependence

$$f(x) \stackrel{x \gg 1}{\longrightarrow} B \ln x \tag{C.7}$$

with some constant B the neglecting of which will only change the slope. A regular and symmetric function with the same long distance behaviour is

$$f(x) = B \sinh^{-1}(x) \tag{C.8}$$

D. Relation between the adiabatic theorem and the Arnold diffusion

An easy explanation for the momentum change $\dot{p} \propto \frac{1}{p}$ along a resonance line can be seen via the adiabatic theorem [GPS62, WS07]. It states that for a Hamiltonian

$$H(\mathbf{p}, \mathbf{q}, a(t)) \tag{D.1}$$

the change of action variables $\mathbf{J}$ tends to zero faster than the change of the time dependent parameter $a(t)$, or more precisely

$$\dot{\mathbf{J}} = \mathcal{O}(\dot{a}^2, \ddot{a}). \tag{D.2}$$

In the case of the Arnold diffusion momentum changes only slightly over one period and therefore approximates the action very well

$$J_x := \oint p_x dx \overset{p_x \gg 1}{\longrightarrow} 2\pi p_x. \tag{D.3}$$

We consider here a particle in a potential $V(x, y, t)$ periodic in all three coordinates. The adiabatic theorem does not hold since the change of the potential is not small. But we can switch to a proper timescale by dividing through one period in x direction $T_x = \frac{2\pi}{p_x}$

$$\tau := \frac{t}{T_x}. \tag{D.4}$$

Neglecting the back action on the y degree of freedom for small perturbations we can consider the potential as dependent on time and x only. On the new

timescale the adiabatic theorem holds

$$\partial_\tau J = \mathcal{O}\left((\partial_\tau a)^2, \partial_\tau^2 a\right) \tag{D.5}$$

and results in the original time in

$$\dot{J} = T_x \mathcal{O}(\dot{a}^2, \ddot{a}) = \frac{2\pi}{p_x}\mathcal{O}(\dot{a}^2, \ddot{a}). \tag{D.6}$$

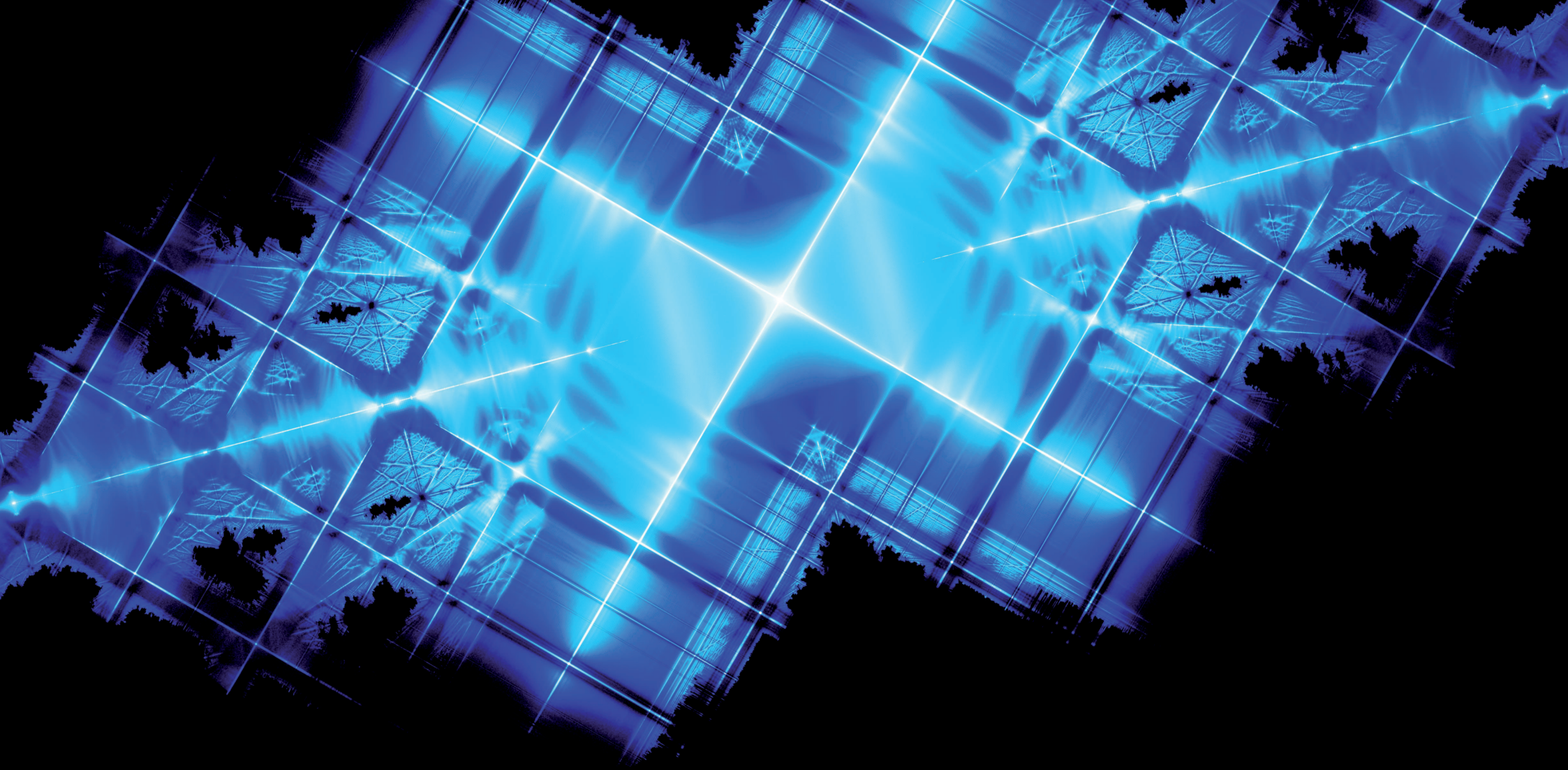
So for initial condition that differ only in the p_x value the change in momentum is

$$\dot{p_x} = \frac{1}{p_x} \tag{D.7}$$

which leads to the logarithmic growth of standard deviation $\sigma_{p_x}(t) \propto \ln t$. The adiabatic theorem constitutes an upper bound and considering the back action of the y DOF will most probably slow it down. So the momentum change will be slower than $\frac{1}{p_x}$.

E. NIM Calendar Cover

One Arnold web received the honor to cover the title and the February of the calendar of 2013 for the Nanosystems Initiative Munich (NIM). I hereby greatfully acknowledge the support of NIM during my phd.



Calendar 2013

Nanosystems Initiative Munich

nim
nanosystems initiative munich

Bibliography

[Amd67] GM Amdahl, *Validity of the single processor approach to achieving large scale computing capabilities*, Proceedings of the April 18-20, 1967, spring joint ... **90** (1967), no. 1, 1–38.

[Arn63] V I Arnold, *Proof of a theorem by A.N. Kolmogorov on the invariance of quasi-periodic motions under small perturbations of the Hamiltonian*, Russian Mathematical Surveys **18** (1963), 9–36.

[Arn64] VI Arnold, *Instability of dynamical systems with several degrees of freedom*, Doklady Akademii Nauk SSSR **156** (1964), no. 1, 9.

[BCM06] Sergio Blanes, Fernando Casas, and Ander Murua, *Symplectic splitting operator methods for the time-dependent Schrodinger equation.*, The Journal of chemical physics **124** (2006), no. 23, 234105.

[BGG84] G Benettin, L Galgani, and A Giorgilli, *Boltzmann's ultraviolet cutoff and Nekhoroshev's theorem on Arnold diffusion*, Nature **311** (1984), 444.

[Blu70] L Bluestein, *A linear filtering approach to the computation of discrete Fourier transform*, IEEE Transactions on Audio and Electroacoustics **18** (1970), no. 4, 451–455.

[BM05] M Borromeo and F Marchesoni, *Noise-assisted transport on symmetric periodic substrates.*, Chaos (Woodbury, N.Y.) **15** (2005), no. 2, 26110.

[Bro28] R Brown, *A brief account of microscopical observations made in the months of June, July and August 1827, on the particles contained in the pollen of plants; and on the general*, Edinburgh new Philosophical Journal **1** (1828), 358 – 371.

[Con02] G. Contopoulos, *Order and Chaos in Dynamical Astronomy*, Springer-Verlag, Berlin, Heidelberg, New York, 2002.

[CT65] J W Cooley and J W Tukey, *An algorithm for the machine calculation of complex Fourier series*, Mathematics of computation **19** (1965), no. 90, 297–301.

[CW90] D. Coppersmith and S. Winograd, *Matrix Multiplication via Arithmetic Progressions*, Journal of symbolic computation **9** (1990), no. 3, 251–280.

[DL93] HS Dumas and J Laskar, *Global dynamics and long-time stability in Hamiltonian systems via numerical frequency analysis*, Physical Review Letters;(United States) **70** (1993), no. 20, 2975–2979.

[DMMFH07] S. Denisov, L. Morales-Molina, S. Flach, and P. Hänggi, *Periodically driven quantum ratchets: Symmetries and resonances*, Physical Review A **75** (2007), no. 6, 063424.

[Don13] J. Dongarra, *Visit to the National University for Defense Technology Changsha, China*, Tech. report, 2013.

[DZFY08] S. Denisov, Y. Zolotaryuk, S. Flach, and O. Yevtushenko, *Vortex and Translational Currents due to Broken Time-Space Symmetries*, Physical Review Letters **100** (2008), no. 22, 224102.

[EH13] C. Efthymiopoulos and M. Harsoula, *The speed of Arnold diffusion*, Physica D: Nonlinear Phenomena **251** (2013), 19–38.

[Fan47] U Fano, *Ionization yield of radiations. II. The fluctuations of the number of ions*, Physical Review **44** (1947), no. 1946, 1–4.

[Fer23] Enrico Fermi, *Beweis, dass ein mechanisches normalsystem im Allgemeinen quasi ergodisch ist*, Physikalische Zeitschrift **24** (1923), 261–265.

[FGL00] C Froeschlé, M Guzzo, and E Lega, *Graphical evolution of the Arnold web: from order to chaos*, Science **289** (2000), no. 5487, 2108–2110.

[FGL05] Claude Froeschlé, Massimiliano Guzzo, and Elena Lega, *Local And Global Diffusion Along Resonant Lines in Discrete Quasi-integrable Dynamical Systems*, Celestial Mechanics and Dynamical Astronomy **92** (2005), no. 1-3, 243–255.

[FMF76] J. A. Fleck, J. R. Morris, and M. D. Feit, *Time-dependent propagation of high energy laser beams through the atmosphere*, Applied Physics **10** (1976), no. 2, 129–160.

[FR90] Etienne Forest and R. D. Ruth, *Fourth-order symplectic integration*, Physica D: Nonlinear Phenomena **43** (1990), 105–117.

[FYZ00] S. Flach, O. Yevtushenko, and Y. Zolotaryuk, *Directed current due to broken time-space symmetry*, Physical review letters **84** (2000), no. 11, 2358–61.

[GL13] Massimiliano Guzzo and Elena Lega, *The numerical detection of the Arnold web and its use for long-term diffusion studies in conservative and weakly dissipative systems.*, Chaos (Woodbury, N.Y.) **23** (2013), no. 2, 023124.

[Gol77] M Goldstein, *Downward price inflexibility, ratchet effects, and the inflationary impact of import price changes: some empirical evidence*, IMF Staff Papers (1977).

[GPS62] H Goldstein, C. P. Poole, and J. L. Safko, *Classical mechanics*, Addison Wesley, San Francisco, 1962.

[GR01] G. Grynberg and C. Robilliard, *Cold atoms in dissipative optical lattices*, Physics Reports **355** (2001), no. 5-6, 335–451.

[HJB85] MT Heideman, DH Johnson, and CS Burrus, *Gauss and the history of the fast Fourier transform*, Archive for History of Exact Sciences **34** (1985), no. 3, 265–277.

[HM09] P. Hänggi and F. Marchesoni, *Artificial Brownian motors: Controlling transport on the nanoscale*, Reviews of Modern Physics **81** (2009), no. 1, 387–442.

[HMWB13] HLRS, Oliver Mangold, Amer Wafai, and Thomas Baumann, *GPU Programming using CUDA, https://fs.hlrs.de/projects/par/events/ 2013/parallel_prog_2013/CUDA1.html*, 2013.

[Hoa62] C.A.R. Hoare, *Quicksort*, The Computer Journal **5** (1962), no. 1, 10–16.

[Int] Intel, *Intel Xeon specifications, http://ark.intel.com/products/64595*.

[Int13] ———, *Using the Intel MPI Library on Intel Xeon Phi Coprocessor Systems, http://software.intel.com/en-us/articles/using-the-intel-mpi-library-on-intel-xeon-phi-coprocessor-systems*, 2013.

[JK10] M Januszewski and M Kostur, *Accelerating numerical solution of stochastic differential equations with {CUDA}*, Computer Physics Communications **181** (2010), no. 1, 183–188.

[KLH09] M. Kostur, J. Luczka, and P. Hänggi, *Negative mobility induced by colored thermal fluctuations*, Physical Review E **80** (2009), no. 5, 051121.

[Kol54] A.N. Kolmogorov, *On the conservation of conditionally periodic motions for a small change in Hamilton's function*, Doklady Akademii Nauk SSSR **98** (1954), 527–530.

[LGF03] Elena Lega, Massimiliano Guzzo, and Claude Froeschlé, *Detection of Arnold diffusion in Hamiltonian systems*, Physica D: Nonlinear Phenomena **182** (2003), no. 3-4, 179–187.

[Lic92] A. J. Lichtenberg, *Arnold diffusion in a torus with time-varying fields*, Physics of Fluids B: Plasma Physics **4** (1992), no. 10, 3132.

[LL92] A.J. Lichtenberg and M.A. Liebermann, *Regular and chaotic dynamics*, Springer, New York, 1992.

[Loc99] Pierre Lochak, *Arnold diffusion; a compendium of remarks and questions*, Hamiltonian systems with three or more degrees of freedom, vol. 533, 1999, pp. 168–183.

[LR01] Jacques Laskar and Philippe Robutel, *High order symplectic integrators for perturbed Hamiltonian systems*, Celestial Mechanics and Dynamical Astronomy **80** (2001), no. 39, 39–62.

[LR09] V. Lebedev and F. Renzoni, *Two-dimensional rocking ratchet for cold atoms*, Physical Review A **80** (2009), no. 2, 023422.

[Mar03] George Marsaglia, *Xorshift RNGs*, Journal of Statistical Software **8** (2003), no. 14, 1–6.

[MO06] Oliver Morsch and Markus Oberthaler, *Dynamics of Bose-Einstein condensates in optical lattices*, Reviews of Modern Physics **78** (2006), no. 1, 179–215.

[Mor02] A. Morbidelli, *Modern celestial mechanics*, Taylor & Francis, London, 2002.

[Mos62] J. K. Moser, *On invariant curves of area perserving mappings of an annulus*, Math. Phsy. Kl. (1962), 1–20.

[MPV86] A Malagoli, Giovanni Paladin, and Angelo Vulpiani, *Transition to stochasticity in Hamiltonian systems: Some numerical results*, Physical Review A **34** (1986), no. 2.

[Nek77] NN Nekhoroshev, *An exponential estimate of the time of stability of nearly-integrable Hamiltonian systems*, Russian Mathematical Surveys **32** (1977), 1.

[Nvi] Nvidia, *Nvidia homepage, http://www.nvidia.com.*

[Nvi13] ———, *CUDA C Programming Guide, http://docs.nvidia.com/cuda/cuda-c-programming-guide/*, 2013.

[OO07] K. Ohmi and K. Oide, *Chaos and emittance growth due to nonlinear interactions in a circular accelerator*, Physical Review Special Topics - Accelerators and Beams **10** (2007), no. 1, 014401.

[PÖ93] J. Pöschel, *Nekhoroshev estimates for quasi-convex Hamiltonian systems*, Mathematische Zeitschrift **216** (1993), 187–216.

[PTVF02] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C++: The Art of Scientific Computing*, Cambridge University Press, 2002.

[SDK05] Holger Schanz, Thomas Dittrich, and Roland Ketzmerick, *Directed chaotic transport in Hamiltonian ratchets*, Physical Review E **71** (2005), no. 2, 026228.

[SDPH11] A. Seibert, S. Denisov, A. V. Ponomarev, and P. Hänggi, *Mapping the Arnold web with a graphic processing unit.*, Chaos (Woodbury, N.Y.) **21** (2011), no. 4, 043123.

[SIC99] J Stenger, S Inouye, and AP Chikkatur, *Bragg spectroscopy of a Bose-Einstein condensate*, Physical Review Letters (1999), no. June, 4569–4573.

[SKH+09] T. Salger, S. Kling, T. Hecking, C. Geckeler, L. Morales-Molina, and M. Weitz, *Directed transport of atoms in a Hamiltonian quantum ratchet.*, Science (New York, N.Y.) **326** (2009), no. 5957, 1241–3.

[SMDS11] J.K. Salmon, M.A. Moraes, R.O. Dror, and D.E. Shaw, *Parallel random numbers: as easy as 1, 2, 3*, Proceedings of 2011

International Conference for High Performance Computing, Networking, Storage and Analysis (2011).

[SMHN04] S. Savel'ev, F. Marchesoni, P. Hänggi, and F. Nori, *Nonlinear signal mixing in a ratchet device*, Europhysics Letters (EPL) **67** (2004), no. 2, 179–185.

[Smi12] Ryan Smith, *NVIDIA Launches Tesla K20 & K20X: GK110 Arrives At Last, http://www.anandtech.com/show/6446/nvidia-launches-tesla-k20-k20*, 2012.

[SSPRG03] M. Schiavoni, L. Sanchez-Palencia, F. Renzoni, and G. Grynberg, *Phase Control of Directed Diffusion in a Symmetric Optical Lattice*, Physical Review Letters **90** (2003), no. 9, 094101.

[Str69] V. Strassen, *Gaussian Elimination is not Optimal*, Numerische Mathematik **13** (1969), 354–356.

[Sup] Supercomputer, *http://de.wikipedia.org/wiki/Supercomputer*.

[Sut05] H. Sutter, *The Free Lunch Is Over : A Fundamental Turn Toward Concurrency in Software*, Dr. Dobb's Journal **30** (2005), no. 3, 1–7.

[Suz76] M. Suzuki, *Generalized Trotter's formula and systematic approximants of exponential operators and inner derivations with applications to many-body problems*, Communications in Mathematical Physics **51** (1976), no. 2, 183–190.

[TK07] Julien Tailleur and Jorge Kurchan, *Probing rare physical trajectories with Lyapunov weighted dynamics*, Nature Physics **3** (2007), no. 3, 203–207.

[TLL79] J.L. Tennyson, A.J. Lichtenberg, and M.A. Liebermann, *Nonlinear Dynamics and the Beam-Beam Interaction*, AIP Conf. Proc.57, 1979, p. 272.

[TRG89] C.C. Tannoudji, J.D. Roc, and G. Grynberg, *Atoms and Photons, Introduction to Quantum Electrodynamics*, Wiley, New York, 1989.

[Tro59] HF Trotter, *On the product of semi-groups of operators*, Proceedings of the American Mathematical Society **10** (1959), 545–551.

[vDU96] von Milczewski J., G. Diercksen, and T. Uzer, *Computation of the Arnol'd web for the hydrogen atom in crossed electric and magnetic fields.*, Physical review letters **76** (1996), no. 16, 2890–2893.

[WP01] H. Wobig and D. Pfirsch, *On guiding centre orbits of particles in toroidal systems*, Plasma physics and controlled fusion **695** (2001).

[WS07] C. G. Wells and S. T. C. Siklos, *The adiabatic invariance of the action variable in classical dynamics*, European Journal of Physics **28** (2007), no. 1, 105–112.

[Yos90] H. Yoshida, *Construction of higher order symplectic integrators*, Physics Letters A **150** (1990), no. 5-7, 262–268.

[ZDH11] V. Zaburdaev, S. Denisov, and P. Hänggi, *Perturbation Spreading in Many-Particle Systems: A Random Walk Approach*, Physical Review Letters **106** (2011), no. 18, 180601.

Abbreviations:

ALU	arithmetic logic unit
API	application programming interface
CPU	central processing unit
CUDA	compute unified device architecture
GPU	graphics processing unit
Flop(Flops)	Floating point operation (per second)

AD	Arnold diffusion
AW	Arnold web
DOF	degree of freedom
MSD	mean square displacement
RMS	root mean square displacement

Danksagung

Mein Dank gilt zuallererst Peter Hänggi für die bereitwillige Aufnahme in den Lehrstuhl. Ich habe von Ihnen nicht nur über die Wissenschaft sondern auch viel über die Gesellschaft gelernt.

Sergey Denisov fühle ich mich verpflichtet für seine Hilfsbereitschaft, seine Offenheit, seinen Ideenreichtum und die philosophischen Debatten.

Gert-Ludwig Ingold danke ich für das Teilen seines breitgefächerten Wissens und sein herausragendes Engagement in der Ausbildung des wissenschaftlichen Nachwuchses, welches auch meinen Horizont erweitert hat.

Mein weiterer Dank gilt Liviu Chioncel für das Zweitgutachten.

Meinen Bürokollegen, Michele Campisi und Antonio Hill, danke ich herzlich. Ich habe viel über eure Arbeitsgebiete und über die Grundlagen der Physik gelernt. Aber vor allem habe ich eure Anwesenheit immer als sehr angenehm empfunden.

Ich danke den Kollegen von Theo1&2 für die produktive und lockere Atmosphäre. Gerhard Schmids erhellende Erläuterungen zur Diffusion müssen an dieser Stelle hervorgehoben werden.

Ralf Utermann danke ich für die Hilfe am Cluster, Alexey Ponomarev für die Unterstützung für das Nobel Laureate meeting.

Sebastian Deffner und Andreas Dechant möchte ich auch für ihre langjährige Freundschaft vom Einmaleins bis zur Quantenthermodynamik danken.

Georg Reuther sei außerordentlicher Dank ausgesprochen für das gemeinsame Studium von Chateau Migraine, circuit-QED und Dialekten.

Des Weiteren danke ich Michael Stark und Martin Wolf für den gemütlichen Mittagsstammtisch.

Ein herzliches Dankeschön richtet sich an Julia Fendt für das Korrekturlesen.

Wir sind immer ein Produkt unseres Umfeldes. Hochmut ist nur angemessen, wenn man ein Umfeld vermisst. Und so will ich zu guter Letzt demütig meiner Familie danken. Ohne euren Rückhalt und Antrieb wäre diese Arbeit nicht zustande gekommen.